Flexible Quantum Kolmogorov-Arnold Networks

Enhancing Quantum KAN Expressivity via Fractional Chebyshev Basis and Trainable QSVT

Bachelor's Thesis Project of

Javier González Otero

Miguel Ángel González Ballester

Mathematical Engineering in Data Science

Year 2024 - 2025

upf. Universitat Pompeu Fabra Barcelona

Escola d'Enginyeria

Acknowledgments

I would like to express my sincere gratitude to Miguel Ángel González Ballester for his supervision and for giving me the opportunity to delve into the world of quantum computing. His support allowed me to connect with other experts in the field, participate in regular meetings with the Quantic group at the Barcelona Supercomputing Center, and present this work at the international conference Quantum Techniques in Machine Learning (QTML).

Adrián Pérez Salinas was also a fundamental pillar throughout this project. His guidance and thoughtful insights inspired some of the most relevant ideas in this work. I am especially grateful for his help and encouragement during the preparation and presentation of this research at QTML.

I would also like to sincerely thank Francesco Aldo Venturelli for his valuable feedback, and support during the first part of the project.

Above all, I wish to express my heartfelt gratitude to my family for their constant support and for believing in me throughout all these years of study. Without them, none of this would have been possible. I am also deeply thankful to my loved ones for their unwavering motivation throughout the development of this thesis, whose presence has been a true source of strength and inspiration.

Abstract

Kolmogorov-Arnold Networks (KANs) gained attention for their expressivity and Multilayer Perceptron (MLP)-level performance. Quantum versions like QKAN use Block-Encodings (BEs) and quantum linear algebra to form nonlinearities as learnable Chebyshev approximations, but lack validated implementations. Others, such as VQKAN or AVQKAN, report results but do not provide open-source code.

In this work, we focus on classical-to-classical QKAN models and present two contributions. First, we introduce the Generalized Fractional order of the Chebyshev Functions (GFCFs) as a classical step prior to QKAN processing. Second, we propose Flex-QKAN, a QKAN architecture with learnable basis functions via trainable QSVT angles. We also release CCQKAN, an open-source QKAN framework [1] for evaluating the proposed methods.

We evaluate our proposed models on three different tasks. The GFCF technique enhances the obtained results; with fixed α, trainability is not affected, while learnable α improves expressivity. Flex-QKAN outperforms CHEB-QKAN in all proposed tasks.

Resum

Les *Kolmogorov-Arnold Networks* (KANs) destaquen per la seva expressivitat i rendiment comparable al dels *Multilayer Perceptron* (MLPs). Les versions quàntiques com QKAN usen *Block-Encodings* (BEs) i àlgebra lineal quàntica per formar no linealitats mitjançant aproximacions de Chebyshev entrenables, però no tenen implementacions validades. Altres implementacions presenten resultats però sense codi obert.

En aquest treball ens centrem en models QKAN que reben i tornen dades clàssiques, presentant dues contribucions. Primer, introduïm les *Generalized Fractional order of the Chebyshev Functions* (GFCFs) com a pas clàssic previ al processament QKAN. Segon, proposem Flex-QKAN, una arquitectura QKAN amb funcions base entrenables mitjançant angles de QSVT parametritzats. També publiquem CCQKAN com a framework basat en QKAN de codi obert[1] per avaluar els mètodes proposats.

Avaluem els nostres models en tres tasques diferents. GFCF aconsegueix millorar els resultats consistentment; amb α fixa l'entrenabilitat no es veu afectada, mentre que amb α entrenable augmenta l'expressivitat. Flex-QKAN millora el CHEB-QKAN en les tasques proposades.

[1] https://github.com/JavierGonzalezOtero02/CCQKAN

Resumen

Las *Kolmogorov-Arnold Networks* (KANs) destacan por su expresividad y rendimiento comparable al de los *Multilayer Perceptron* (MLPs). Las versiones cuánticas como QKAN usan *Block-Encodings* (BEs) y álgebra lineal cuántica para formar no linealidades mediante aproximaciones de Chebyshev entrenables, pero carecen de implementaciones validadas. Otras implementaciones presentan resultados aunque sin código abierto.

En este trabajo nos centramos en modelos QKAN que reciben y devuelven datos clásicos, presentando dos contribuciones. Primero, introducimos las *Generalized Fractional order of the Chebyshev Functions* (GFCFs) como paso clásico previo al procesamiento QKAN. Segundo, proponemos Flex-QKAN, una arquitectura QKAN con funciones base entrenables mediante ángulos de QSVT parametrizados. También publicamos CCQKAN como framework basado en QKAN de código abierto[2] para evaluar los métodos propuestos.

Evaluamos nuestros modelos en tres tareas distintas. GFCF consigue mejorar los resultados consistentemente; con α fija la entrenabilidad no se ve afectada, mientras que con α entrenable aumenta la expresividad. Flex-QKAN mejora a CHEB-QKAN en las tareas propuestas.

[2]https://github.com/JavierGonzalezOtero02/CCQKAN

Preface

Throughout my studies, I've been drawn to the intersection of mathematics and technology. In recent years, I discovered Machine Learning (ML) and became particularly interested in its potential. During a Deep Learning course, I came across Kolmogorov-Arnold Networks (KANs), and was fascinated by their self-designing capabilities, eventually dedicating my final project to this topic.

Simultaneously, I began exploring quantum computing and its potential applications in ML, which felt like the perfect combination. When I saw that Miguel Ángel González Ballester was offering a project in this area, I eagerly proposed studying KANs in the quantum environment.

During my research, I found several proposals, and one in particular stood out: *Quantum Kolmogorov-Arnold Networks* (QKAN), which offered a mathematically detailed construction. However, while some other implementations reported results, most did not release their code publicly.

This work presents two proposals to enhance QKAN flexibility: using Generalized Fractional-order Chebyshev Functions (GFCFs) and introducing trainable angles in the QSVT procedure. The goal is to improve the expressivity of QKAN-based models while still keeping a strong connection to the original theoretical ideas.

The contributions of this project have been submited to the international conference Quantum Techniques in Machine Learning (QTML).

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

This chapter introduces the main topics of the research and serves as a starting point for readers familiar with linear algebra, computer science, and Machine Learning (ML).

It begins with an overview of quantum computing, followed by an introduction to Quantum Machine Learning (QML). Then, it presents Kolmogorov-Arnold Networks (KANs) and concludes with the presentation of Quantum Kolmogorov-Arnold Networks.

1.1 Quantum computing

Quantum computing is a computational paradigm based on the laws of quantum mechanics. Unlike classical systems, where the fundamental unit of information is the bit [5], quantum systems use the qubit [5, 6, 7].

1.1.1 Qubit overview

A classical bit is described by exactly one of two possible states, 0 or 1. Therefore, a classical system composed of n bits can only be in one of the 2^n possible states at any given time.

In contrast, a qubit state can be represented as a linear superposition of two orthonormal basis states. Upon measurement (observation), the state collapses probabilistically to one of the basis states [8]. The most common set of basis states is called *computational basis* and its components are $|0\rangle = (1, 0)^T$ and $|1\rangle = (0, 1)^T$. The computational basis states serve as quantum analogues of the classical 0 and 1. Equation 1.1 gives a mathematical representation of a single-qubit state with respect to the computational basis.[6].

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle := (\alpha, \beta)^T \tag{1.1}$$

where $\alpha, \beta \in \mathbb{C}$, and $|\alpha|^2 + |\beta|^2 = 1$ ensuring the state is normalized. Complex coefficients α and β are called probability amplitudes, intuitively, they represent the contribution of each basis state to the overall qubit state $|\psi\rangle$. According to the Born rule [9], which will be explained in detail in the next section, the squared modulus $|\alpha|^2$ and $|\beta|^2$ represent the probabilities of measuring the qubit in the states $|0\rangle$ and $|1\rangle$ respectively.

Besides the computational basis, there exist other important sets of basis states. For example the Hadamard basis states (also called X-basis states), defined in terms of the computational basis states as $|+\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$, $|-\rangle = \frac{|0\rangle-|1\rangle}{\sqrt{2}}$. Similarly, the Y-basis, defined by $|+i\rangle = \frac{|0\rangle+i|1\rangle}{\sqrt{2}}$ and $|-i\rangle = \frac{|0\rangle-i|1\rangle}{\sqrt{2}}$.

Bloch Sphere representation

The state of a single qubit can also be visualized on the Bloch sphere [10], as shown in Figure 1.1. This representation allows us to interpret quantum state transformations as rotations.

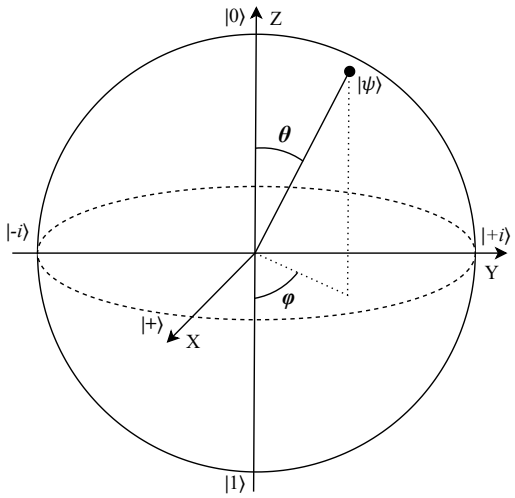


Figure 1.1: Bloch sphere representation of the state $|\psi\rangle$ of a single qubit system. The state $|\psi\rangle$ can be written as $|\psi\rangle = \cos\frac{\theta}{2}\,|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}\,|1\rangle$.

Multi-qubit states

Quantum systems are not restricted to a single qubit. Multi-qubit quantum systems exist, and if they are *separable* [11], their joint state is represented by the tensor product of the individual qubit states. This means that a separable n-qubit system formed by qubits $q_1, \ldots, q_n$ with states $|\psi\rangle_{q_1}, \ldots, |\psi\rangle_{q_n}$, has its qubits prepared separately and is described by the state in Equation 1.2.

$$|\Psi\rangle = |\psi\rangle_{q_1} \otimes \ldots \otimes |\psi\rangle_{q_n} \tag{1.2}$$

If this representation is not possible, the state is called *entangled* [11]. By considering the representation in 1.2, basis states can be redefined to multi-qubit systems, for instance, the computational basis states of an n-qubit system are formed by the orthonormal set of states $\{|j\rangle\}_{j=0}^{n-1}$, where $|j\rangle$ represents the tensor product of $|0\rangle$ and $|1\rangle$ states that correspond to the binary representation of the integer j. That is, $|j\rangle = |b_0 b_1 \cdots b_{n-1}\rangle$ with $b_i \in \{0, 1\}$ and $j = \sum_{i=0}^{n-1} b_i 2^i$. For example, considering a 4-qubit system, $|2\rangle = |0\rangle \otimes |0\rangle \otimes |1\rangle \otimes |0\rangle = |0010\rangle$.

Quantum entanglement

Entangling is a central property of Quantum Systems, as it allows to define states that would be impossible to define otherwise [11]. Entangled states [11], are multi-qubit quantum states in which their qubits have been prepared jointly, which may lead to the impossibility of writing them by separating the state of each qubit as it would be done by terms of Equation 1.2 [11].

An example of this is the Bell state, described in Equation 1.3

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{1.3}$$

which cannot be written as a tensor product of two single-qubit states.

Multi-Qubit states: general form

The n-qubit state $|\Psi\rangle$ is described by a linear superposition of 2^n basis states:

$$|\Psi\rangle = \sum_{j=0}^{2^{n-1}} a_j \, |j\rangle \tag{1.4}$$

Where $|j\rangle$ is the j-th basis state of the n-qubit computational basis, $a_j \in \mathbb{C}$ are the probability amplitudes and therefore $\sum_{j=0}^{2^{n-1}} |a_j|^2 = 1$.

By using the representation in Equation 1.4, the Bell state is written as: $|\Phi^+\rangle = \frac{1}{\sqrt{2}}|00\rangle + 0|01\rangle + 0|10\rangle + \frac{1}{\sqrt{2}}|11\rangle$ with probability amplitudes $a_0 = \frac{1}{\sqrt{2}}$, $a_1 = 0$, $a_2 = 0$ and $a_3 = \frac{1}{\sqrt{2}}$.

The state of a system formed by n qubits resides in a Hilbert space whose dimensionality grows exponentially with the number of qubits n. [6, 12]. This property reveals one of the key advantages of quantum computing: an n-qubit quantum system can process all 2^n possible basis states simultaneously, but the classical system can only process one of the 2^n possible states [6].

1.1.2 Fundamental operations on quantum systems

There are two fundamental types of operations that can be performed on an n-qubit quantum system: state transformations and measurements. Quantum circuits can be built by combining these two types of operations [6].

State Transformations

Quantum state transformations are implemented using quantum logic gates which are often referred to as *unitaries* because they are represented by parametrized unitary matrices [4]. A unitary U can act on $k \in [1, n]$ qubits, and is called k-qubit unitary. The dimension of U is $2^k \times 2^k$.

Single-qubit unitaries act on one qubit and represent rotations along the Bloch-Sphere according to some angle parameters. The most common single qubit gates are $R_z(\theta)$, $R_x(\alpha)$ and $R_y(\beta)$, which apply rotations around the z, x and y axis of the Bloch-Sphere respectively. Their matrix definitions are shown in Equation C.1 [4]. Pauli matrices are special cases of these unitaries where the rotation angle is π, they allow to invert the state of a qubit along a particular axis, these matrices are represented by $Z = R_z(\pi)$, $X = R_x(\pi)$ and $Y = R_y(\pi)$ [13].

A universal single-qubit rotation gate can be constructed from the previous set of gates using the Euler decomposition $U(\theta, \phi, \lambda) = R_z(\phi)R_y(\theta)R_z(\lambda)$ [14].

Multi-qubit unitaries act on more than one qubit. Some produce separable states and can be written as tensor products of single-qubit gates. For example, a 2-qubit gate acting on a 2-qubit state $|\psi\rangle$ can be defined as $U_\psi = U_0(\theta_0, \phi_0, \lambda_0) \otimes U_1(\theta_1, \phi_1, \lambda_1)$ leading to a final state $|\Psi\rangle = U_\psi|\psi\rangle$ [15].

In contrast, entangling gates cannot be expressed as tensor products of single-qubit gates. An important example of this is the $CNOT$ gate [8], defined in Equation C.2. This 2-qubit gate applies an X gate to a target qubit if a control qubit is in state $|1\rangle$. Table D.1 shows the action of a $CNOT$ gate applied to different 2-qubit computational basis states [6]. The definition of the $CNOT$ gate introduces the concept of controlled and multi-controlled gates, which apply an arbitrary unitary to a set of target qubits depending on the value of one or more

qubits respectively. Control qubits can also be in state $|0\rangle$ to activate the action of the unitary on the target qubits.

Measurements

Measurements, on the other hand, correspond to the process of extracting information from a quantum system by observing it. Physically, this involves determining the value of system properties named observables. Classical observables are the position, speed, momentum... . Examples of quantum-observables are the energy of a system, the spin projection of an electron along the z-axis or the polarization of photons. [16, 17]. To extract the value of an observable from a system, a quantum test must be performed [16]. Quantum tests consist in two phases. A state preparation phase in which all necessary state transformations are applied to the system, and the measuring phase, in which a real-valued label associated to the value of the observable being measured is returned [16]. In quantum systems, an observable $\hat{O}$ is mathematically represented by a single-qubit unitary matrix, that is, a single-qubit quantum gate [16].

The possible outcomes of a quantum test are the eigenvalues of the observable $\hat{O}$ being measured. According to the Born rule, the probability of obtaining a particular eigenvalue λ upon measurement is given by $Pr(\lambda) = \langle\psi| \hat{\Pi}_\lambda |\psi\rangle$ where $\hat{\Pi}_\lambda$ is the projector operator over the subspace of eigenvectors associated to λ and $|\psi\rangle$ is the state of the system prior to measurement [9, 18]. If the dimension of $\hat{\Pi}_\lambda$ is 1, then there is only one associated eigenvector to λ, and hence $P(\lambda) = |\langle i|\psi\rangle|^2$ where $\hat{O}|i\rangle = \lambda|i\rangle$ [9].

Supposing that λ is the measured eigenvalue, the state $|\psi\rangle$ of the quantum system collapses to $|\psi'\rangle = \frac{\hat{\Pi}_\lambda|\psi\rangle}{\langle\psi|\hat{\Pi}_\lambda|\psi\rangle}$ which is the projection of $|\psi\rangle$ into $\hat{\Pi}_\lambda$ [18].

If a deterministic value needs to be drawn from the system, one can obtain the expected value of measuring an observable $\hat{O}$ in a system with state $|\psi\rangle$. This is denoted as $\langle\hat{O}\rangle_\psi$ and is mathematically defined as $\langle\hat{O}\rangle_\psi = \langle\psi|\hat{O}|\psi\rangle$ [16].

Quantum Circuits

A quantum circuit represent sequences of quantum operations [19]. In a quantum circuit, the evolution of the state of each qubit is followed by what is called *wires* [20], represented by a straight line in the graphical representation of the circuit [4]. Quantum gates are represented by a wide variety of symbols depending on the action of the gate. A summary of gates graphical representation can be found in Figure B.1 [4].

In a quantum circuit, it is possible to measure a subset of the qubits while continuing to apply quantum gates to the remaining ones. For this reason, the measurement operation is often represented graphically by a measurement gate,

as illustrated in Figure B.1 [21]. An example of a quantum circuit is shown in Figure 1.2.

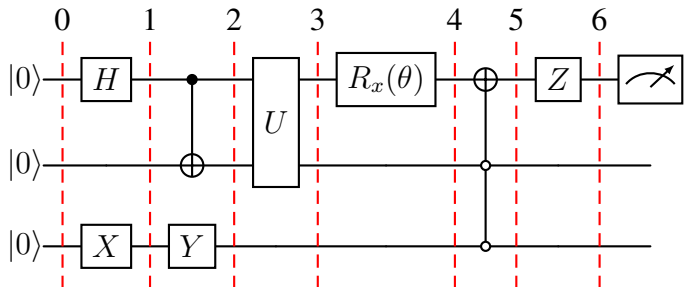


Figure 1.2: Graphical representation of a quantum circuit formed by 3 wires. It is initialized to state $|\psi\rangle = |0\rangle \otimes |0\rangle \otimes |0\rangle$ and returns either $|0\rangle$ or $|1\rangle$ measured on the first qubit. The circuit is divided in 7 stages that can be considered individually: Stage 0 corresponds to initialization, Stages 1 through 6 represent successive gate applications, and the final stage corresponds to measurement.

A quantum circuit can be represented by a single unitary matrix acting on all the qubits of the system. To obtain the matrix representation of the circuit, it is necessary to introduce the concept of stages, which can be interpreted as the main division points of a quantum circuit. Stages act as complete quantum circuits, hence they also have a matrix representation formed by the tensor product of all gates acting within the stage. Figure 1.2 shows in red vertical dashed lines the separation between each stage.

Then, to obtain the matrix representation of the full circuit, all matrix representations of the stages forming the circuit are multiplied together [19, 20].

It is important to notice that any n-qubit quantum gate formed by a combination of simpler quantum gates acting on $n' \leq n$ qubits is also a quantum circuit itself.

Parametrized quantum circuits and ansatzes

Figure 1.2 contains an $R_x(\theta)$ gate in stage 3 - 4. Since that gate depends on a rotation angle parameter, it is classified as a parametrized gate. Consequently the circuit is considered a parametrized quantum circuit (PQC) [22] with input parameter θ. The value of the measurement at the end of the circuit will depend on θ.

A specific circuit design aimed at solving a particular task is typically referred to as an *ansatz* [23]. An ansatz does not necessarily need to be completely fixed, it can be a PQC, which uncovers a powerful analogy between PQCs and classical

ML models: a model, (ansatz for the quantum case), is defined, and its parameters are optimized according to a learning strategy.

1.2 Quantum Machine Learning

When the quantum principles discussed in the previous section are applied to learning systems, the resulting field is known as Quantum Machine Learning (QML). It explores how quantum resources can be used to enhance the performance of learning algorithms [24].

Data in quantum systems

A necessary distinction in QML has to do with the nature of the data. Data generated from classical systems is referred to as classical data [7]. It typically corresponds to real-valued vectors in $\mathbb{R}^n$, encoded in bits. In contrast, quantum systems process information, known as quantum data, through quantum states.

To process classical data in a quantum system it must be first transformed to quantum data. This is done by encoding it into the quantum system through a process called *quantum data encoding* [7].

Among the many existing methods for quantum data encoding, two widely used approaches are: *amplitude encoding* [7], where classical data is encoded into the probability amplitudes of a quantum state; and *angle encoding* [7], in which classical data is embedded as parameters of a parameterized quantum circuit (PQC).

Categories of QML algorithms

Depending on how classical and quantum components interact, QML algorithms could be categorized as:

- **Classical algorithms**. This corresponds to the traditional machine learning setup. Data and processing systems involved are classical. Neural networks, linear regression, k-means, are examples of learning algorithms belonging to this category.

- **Hybrid quantum-classical algorithms**. Quantum and classical processing coexist during the training process. Variational Quantum Algorithms (VQAs) [25] are an example of this.

  In a VQA, classical data is processed through a PQC. Then a deterministic classical value is extracted from the circuit, this value is typically the

expected value of an observable. Finally, learnable parameters (which are formed by classical data) are optimized through classical computing.

As it will be revealed in the following sections, this is the most relevant set of QML algorithms for this project purposes.

- **Quantum algorithms**. Only quantum systems are used for the training process where both the data and learning process are entirely quantum. Quantum phase estimation is an algorithm belonging to this setup [26].

1.3 Kolmogorov-Arnold Networks

Returning to classical systems, Kolmogorov-Arnold Networks (KANs) [27] are a novel type of neural network. Unlike traditional neural networks, which are based on the Universal Approximation Theorem (UAT) [28], KANs are inspired by the Kolmogorov-Arnold representation Theorem (KAT) [29] and hence, consist solely of parametrized activation functions and their summation. Their objective is to offer much more interpretable models than MLPs.

1.3.1 Kolmogorov-Arnold representation Theorem

KANs are inspired by KAT, which states that if a multivariate function $f(x_1, \ldots, x_n)$ is continuous on a bounded domain, then it can be expressed as a finite composition of univariate continuous functions and the binary operation of summation. Specifically:

$$f(\mathbf{x}) = f(x_1, \ldots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right) \tag{1.5}$$

where $\phi_{q,p} : [0,1] \to \mathbb{R}$ and $\Phi_q : \mathbb{R} \to \mathbb{R}$ are arbitrarily complex continuous functions. This suggests that models based on KAT may exhibit high expressiveness, as they are capable of learning data-specific nonlinearities directly. As a first attempt of converting this theorem to a Neural Network, one could consider every summation of univariate continuous functions as a layer. In this scenario, KAT is restricted to a two-layer model with $\mathcal{O}(n^2)$ functions in total.

1.3.2 KAN architecture

KANs provide an empirical generalization of KAT by allowing an arbitrary number of layers, as well as a flexible number of functions within each layer.

KAN layer

KAN layers can be defined by the $N^{(l)}$-dimensional vector:

$$\Phi^{(l)} = \left(\sum_{p=1}^{N^{(l-1)}} \phi_{p,1}^{(l)}(x_p), \ldots, \sum_{p=1}^{N^{(l-1)}} \phi_{p,N^{(l)}}^{(l)}(x_p) \right) \tag{1.6}$$

where $l \in [1, L]$ is the index of the layer, $\phi_{pq}^{(l)}$ is the learnable activation function between input dimension $p \in [1, N^{(l-1)}]$ and the output dimension $q \in [1, N^{(l)}]$, and $\Phi^{(0)} = \mathbf{x}$.

Stacking KAN layers

KAN layers can be stacked to form multi-layered models. This is done by composition:

$$\mathrm{KAN}(\mathbf{x}) = \Phi^{(L)} \circ \ldots \circ \Phi^{(1)}(\mathbf{x}) \tag{1.7}$$

KANs architecture can be summarized in a list of input sizes for each layer (including layer 0 corresponding to the dimension of $\mathbf{x}$): $[N^{(0)}, \ldots, N^{(L)}]$ Figure1.3 shows a graphical representation of a KAN architecture.

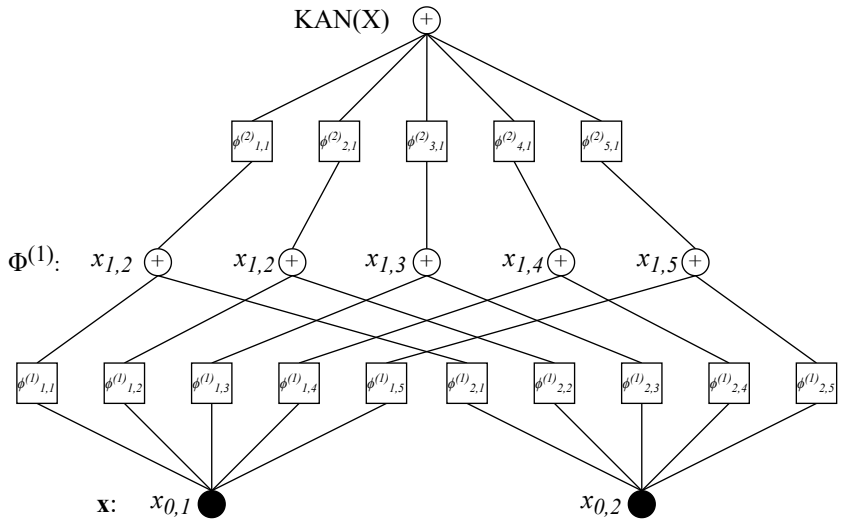


Figure 1.3: KAN architecture with $L = 2$ layers, $N^0 = 2$ (input layer does not compute in the total number of layers), $N^{(1)} = 5$ and output dimension $N^{(2)} = 1$. This architecture can be summarized in a list: $[2, 5, 1]$

1.4 Quantum Kolmogorov-Arnold Networks

Quantum KANs [30, 1, 2, 31] are a recent class of QML algorithms considered to be the direct quantum counterpart of classical Kolmogorov-Arnold Networks. Several approaches have been proposed for constructing quantum versions of KANs [30, 1, 2, 31],but these are still at an early stage and have not yet been peer reviewed. Therefore, they remain part of the current state-of-the-art in Quantum KAN architectures and will be discussed in detail in Chapter 2.2.1.

Chapter 2

STATE OF THE ART

This chapter provides the necessary context for understanding the current development situation of Quantum KANs. It starts by outlining the principal achievements and results of classical KANs, and then moves to discuss the current landscape of Quantum KANs, offering a high-level overview of the most relevant architecture proposals and, if available, their results.

2.1 Current State of KANs

Several ML algorithms based on KAT existed prior to the introduction of KANs [32, 33], however, most of them struggled to accurately represent complex functions [27].

2.1.1 KANs are accurate and interpretable

KANs [27] were introduced in May 2024. As previously stated, the main contribution of KAN was to empirically generalize KAT by allowing general depths and widths. This modification of KAT allowed KANs with a limited number of layers to represent arbitrarily complex functions in an interpretable way. This achievement attracted considerable attention from the ML community and several publications were made based on KANs [34, 35, 36] including this project itself.

Although missing an explicit mathematical demonstration on the universality of KAT generalization, KANs have been empirically shown to be at least as accurate as Multi Layer Perceptrons (MLPs) in symbolic regression tasks, using less parameters and offering greater interpretability as shown in Figure 2.1 [27, 36]. Accepting deeper constructions than those suggested by KAT, improves KANs expressive power [32], exactly as in the MLP case. In fact, KANs have shown to be at least as expressive as MLPs [37].

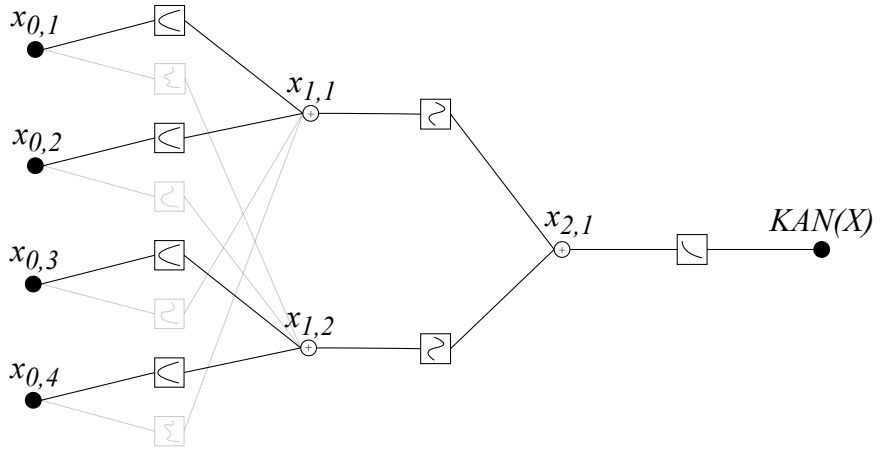


Figure 2.1: [4, 2, 1, 1] KAN aimed to solve a fitting problem where dataset has been synthetically generated by $e^{sin(x_1^2+x_2^2)+sin(x_3^2+x_4^4)}$. Squared boxes represent the learned univariate functions; their opacity represents their relative contribution to the final prediction.

However, the authors state that the primary objective of KANs is not to compete with MLPs, but rather to offer enhanced interpretability.

2.1.2 B-Splines: local but slow

In the original KAN implementation, the authors proposed using B-splines [38] as the learnable activation functions. B-splines are local, meaning that changes at one point do not affect the function globally. This locality is particularly advantageous for achieving fine-grained adaptability during training. However, this choice also introduces computational overhead.

The main drawback of KANs lies in their slow training process. When constructed according to the original design, a KAN with the same number of parameters as an MLP is, on average, up to 10 times slower to train [27]. A key reason for this inefficiency is that each connection in a KAN learns a distinct activation function, which results in non-uniform partial derivatives throughout the network. This variation reduces the effectiveness of parallel computation during backpropagation. Nevertheless, more efficient implementations have since emerged, substantially reducing training times [39, 40].

2.1.3 Applications of KANs

The effectiveness of KANs and their variations has been studied across a wide range of machine learning domains, including computer vision [34], time series analysis [35], and reinforcement learning [41]. As previously discussed, their key advantage is their interpretability, which makes them particularly suitable for contexts where standard MLPs fall short, for example in clinical applications or scientific problems where the objective is to uncover an underlying function modeling the relationships in the data. Across most applications, researchers have reported results consistent with the original authors' findings: KANs offer high interpretability and comparable accuracy to MLPs, often with significantly fewer parameters, albeit at the cost of a slower training process.

2.2 KAN and QML

The resonance of KANs reached the QML community, and by June 2024, usability of KANs had already been explored in quantum contexts [42], and a preliminary quantum version was proposed [1]. However, the state of the art in Quantum KANs is still mostly theoretical.

To the best of my knowledge, a total of 4 quantum versions of KANs have been proposed by today:

Name	Abbreviation
Quantum Kolmogorov-Arnold Networks [30]	QKAN
Variational Quantum Kolmogorov-Arnold Networks [1]	VQKAN
Adaptive Variational Quantum Kolmogorov-Arnold Networks [2]	AVQKAN
Enhanced Variational Quantum Kolmogorov-Arnold Networks [3]	EVQKAN
Quantum Kolmogorov-Arnold Networks by combining quantum signal processing circuits [31]	–

Table 2.1: Summary of quantum Kolmogorov-Arnold network variants and their abbreviations.

Last version, only provides a high-level theoretical explanation of a possible construction, lacking concrete implementation details and results. For that reason, this project focuses on QKAN, VQKAN, Adaptive VQKAN and EVQKAN, which will be introduced below.

2.2.1 QKAN

QKAN [30] was introduced in October 6 2024. Authors have not published any implementation, nonetheless, the authors provide a detailed mathematical explanation and resource analysis of a particular construction of QKAN named CHEB-QKAN. QKAN [30] is a quantum algorithm that relies on quantum linear algebra [43] to construct a circuit whose matrix representation encodes the output vector of a classical KAN (Stated in Equation 1.7) within a diagonal sub-block. The architecture of QKAN is constructed by recursively stacking quantum circuits, each corresponding to a layer analogous to those in a classical KAN.

QKAN and Block-Encodings

The method is based on Block-Encodings (BE) [44, 45] (defined in Appendix A.1). In a nutshell, BEs are $(n+a)$-qubit unitaries that encode an arbitrary matrix $A_{N\times N}$, where $n = \lceil log_2(N) \rceil$ and a is the number of anzilla qubits needed, up to a factor α with precision ε. This means that if A_{BE} is an (α, a, ε)-BE of A, its matrix form is represented as:

$$A_{BE} = \begin{bmatrix} \frac{A'}{\alpha} & * \\ * & * \end{bmatrix} \tag{2.1}$$

where $\|\alpha A' - A\| \leq \varepsilon$. A BE is said to be *diagonal* if the matrix encoded is diagonal. QKAN uses diagonal BEs as input and output, therefore, within its construction, the terminology BE will always refer to a diagonal BE.

QKAN methodology

Assuming that input (diagonal) BEs have been defined, QKAN applies transformations to them through quantum linear algebra to obtain BEs of the intermediate vectors featured in a classical KAN, and finally, an output BE corresponding to the output of a KAN is obtained. Independently of QKAN, this resulting BE can then be applied to a quantum state to either measure the system to produce a classical output or just keep the resulting state in a quantum register.

In QKAN, activation functions between nodes on two layers are implemented by parametrized linear combinations of polynomials acting as basis functions. BEs of this polynomials are constructed through the Quantum Singular Value Transformation (QSVT) [45, 46] framework (explained with detail in Appendix A.3). In a nutshell, QSVT allows to apply polynomial transformations to the singular values of matrices encoded as BEs. Then, the output BEs after application of QSVT are linearly combined with learnable weights through the Linear Combination of Unitaries (LCU) (explained in Appendix A.4) algorithm. LCU, allows

to form weighted linear combinations of unitaries. Then, BEs of the nonlinearities are again linearly combined to form the output BE of each layer, corresponding to the summation operation in a classical KAN.

The BE of a layer can then be used as input BE for the next layer, enabling a recursive construction, that ends with the formation of the BE corresponding to the output of a QKAN, denoted as U_{QKAN}.

If a classical deterministic output needs to be extracted, the output of a QKAN can be used as the principal block of a Hadamard Test [47] (defined in Appendix A.5), which allows the estimation of each of the diagonal values in the BE.

Authors propose a particular construction of QKAN named CHEB-QKAN.

CHEB-QKAN

In CHEB-QKAN, authors propose to use the Chebyshev polynomials of the first kind [48] (defined in Appendix A.6) as the set of basis functions used to construct the learnable activation functions.

Chebyshev polynomials of the first kind, from now on, simply Chebyshev polynomials, form a basis of the functions space. They are defined as $T_r(x) = \cos(r \arccos(x))$, where r is the degree of the polynomial constructed. As they are a basis of the function space, its linear combination can approximate any function with arbitrary precision.

CHEB-QKAN forms its activation functions as linear combinations of Chebyshev polynomials. Although the original authors did not provide a specific notation for the output BE of a CHEB-QKAN, for clarity and consistency within this work, it will be denoted as U_{CHEB}.

Each layer of the CHEB-QKAN model constructs a unitary U_Φ, which serves as the input BE for the subsequent layer. The following outlines the five key steps involved in constructing a single CHEB-QKAN layer.

CHEB-QKAN layer

Input model Supposing we are constructing the first layer of a CHEB-QKAN, the input is a $(1, a_x, \varepsilon_x)$-BE denoted as U_x of a diagonal matrix that encodes an N-dimensional vector in the $[-1, 1]^n$ domain. This vector may correspond to classical or quantum data (state amplitudes). The input BE can be constructed in many different ways [49, 44, 43]. The layer is said to have N input nodes indexed by $p \in [0, N-1]$.

Step 1: Dilate Supposing that the layer has K output nodes indexed as $q \in [0, K-1]$, a total of NK parametrized activation functions will be constructed between each input and output node. To allocate this, the proposed idea is to

expand the input Block Encoding U_x by appending $k = log_2(K)$ qubits. This operation is equivalent to perform $U_x \otimes I_k$, which will be a $(1, a_x, \varepsilon_x)$-BE of the diagonal matrix:

$$\text{diag}(\underbrace{x_1, \ldots, x_1}_{K}, \underbrace{x_N, \ldots, x_N}_{K}).$$

which contains K copies of each input element.

This unitary can be also seen in quantum circuit terms. This is shown in Figure 2.2.

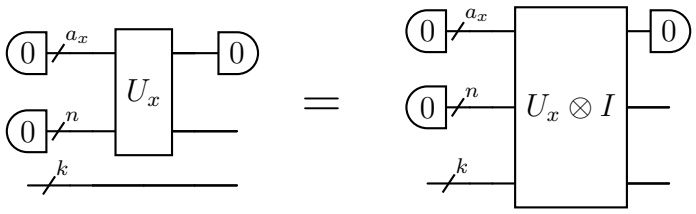


Figure 2.2: Circuit implementing the DILATE step for constructing $U_x \otimes I$

Step 2: CHEB This step implements the Chebyshev basis functions via QSVT. In this process U_x and $U_x^\dagger$ are interleaved by Projector-Controlled Phase (PCPhase)-shift operators (defined in Appendix A.2) with fixed angle parameters. As U_x is a diagonal $(1, a_x, \varepsilon_x)$-BE, the singular values of the encoded matrix are the diagonal values themselves. Hence, applying QSVT by taking as input the $U_x \otimes I_k$ unitary from the last step, a $(1, a_x, \varepsilon)$-Block Encoding of the diagonal matrix

$$\text{diag}\left(\underbrace{T_r(x_1), \ldots, T_r(x_1)}_{K}, \ldots, \underbrace{T_r(x_N), \ldots, T_r(x_N)}_{K}\right) \tag{2.2}$$

can be obtained. The resulting BE is denoted as U_{T_r}. And the circuit for this step is shown in Figure 2.3.

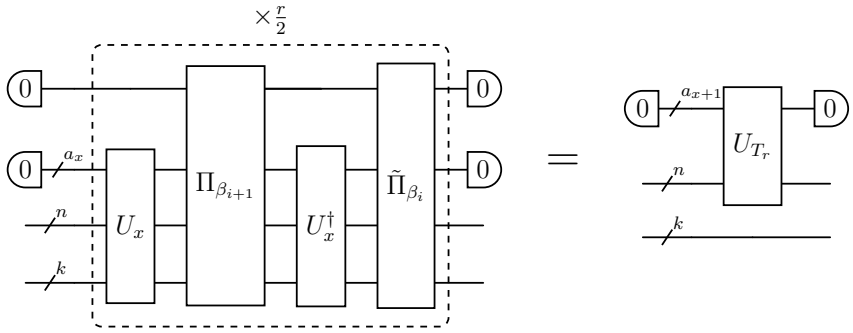


Figure 2.3: Circuit implementing the CHEB step and its equivalence with the U_{T_r} unitary. U_x is the input BE. Π_{β_i} and $\tilde{\Pi}_{\beta_i}$ are the PCPhase-shift operators, where $\beta_0 = (1-r)\frac{\pi}{2}$ and $\beta_{i>0} = \frac{\pi}{2}$ with $i \in [0, r-1]$. An anzilla qubit is added in this step.

This step must be repeated d times to form $U_{T_1} \dots U_{T_d}$ BEs that will be combined in the next step to form the activation functions. Since $T_0(x) = 1$, U_{T_0} is formed by the Identity operator. d is a user-specified parameter referring to the maximum approximation degree of the Chebyshev expansion that forms the activation functions.

Step 3: MUL This step corresponds to obtaining each of the weighted coefficients that are summed-up to build the activation functions. As there are a total of $d+1$ coefficients for every activation function between an input node p and output node q, a total of $(d+1)NK$ learnable weights are needed. These weights are introduced to the circuit as $d+1$ $(1, a_w, \varepsilon_w)$-BEs, noted as $U_w^{(r)}$, of diagonal vectors with size NK. In the end, this step must be repeated $(d+1)$ times, where for each r, a $\big(1, a_x + 1 + a_w,\ 4r\sqrt{\varepsilon_x} + \varepsilon_w\big)$-BE, denoted as $U_w U_{T_r}$ of the following diagonal matrix is constructed:

$$\text{diag}\big(\underbrace{w_{11}^{(r)} T_r(x_1), \dots, w_{1K}^{(r)} T_r(x_1)}_{K}, \dots, \underbrace{w_{N1}^{(r)} T_r(x_N), \dots, w_{NK}^{(r)} T_r(x_N)}_{K}\big) \tag{2.3}$$

This is achieved by directly multiplying U_{T_r} and $U_w^{(r)}$. The circuit that implements $U_w U_{T_r}$ of this step is shown in Figure 2.4

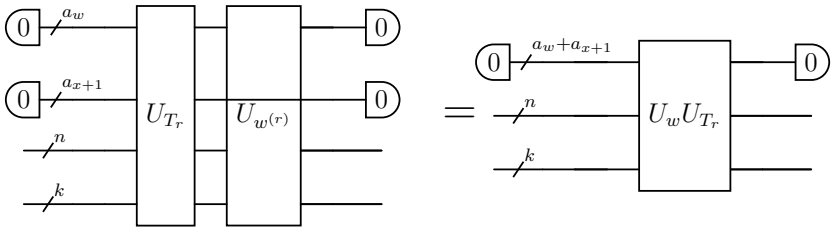


Figure 2.4: Circuit implementing the MUL step for constructing $U_w U_{T_r}$

Step 4: LCU In this step, each of the NK activation functions are constructed by linearly combining the previous $d+1$ BEs for every pair of input and output nodes. This is done through the LCU algorithm (explained in detail in Appendix A.4), and in the end, this step forms a $\big(1, a_x + 1 + a_w + \lceil log_2(d+1) \rceil,\ 4r\sqrt{\varepsilon_x} + \varepsilon_w\big)$-BE, denoted as U_ϕ, of the diagonal matrix

$$\text{diag}\left(\underbrace{\phi_{11}(x_1), \ldots, \phi_{1K}(x_1)}_{K}, \ldots, \underbrace{\phi_{N1}(x_N), \ldots, \phi_{NK}(x_N)}_{K}\right) \tag{2.4}$$

, where each $\phi_{pq}(x_p)$ is the learnable activation function between input p and output q, and is defined as follows:

$$\phi_{pq}(x) = \frac{1}{d+1} \sum_{r=0}^{d} w_{pq}^{(r)} T_r(x_p) \tag{2.5}$$

To this end, a $\lceil log_2(d+1) \rceil$-qubit state is prepared in equal superposition through Hadamard gates. Then, each $U_w U_{T_0} \ldots U_w U_{T_d}$ is controlled by the $\lceil log_2(d+1) \rceil$ anzilla wires added to be summed together. Finally, the $\lceil log_2(d+1) \rceil$-qubit state is returned to the computational basis applying Hadamard gates. The circuit implementing this step is shown in Figure 2.5.

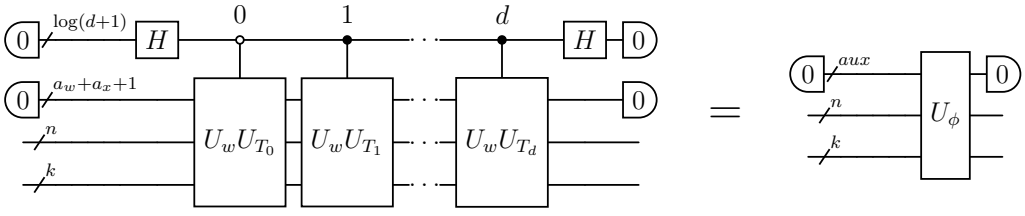


Figure 2.5: Circuit implementing the LCU step for constructing U_ϕ. aux congregate all anzilla qubits and its value is $aux = \log(d+1) + a_w + a_x + 1$. Control values are $|r\rangle$.

Step 5: SUM This is the final step of the construction of the CHEB-QKAN layer. Its objective is to build a $\left(1, a_x + 1 + a_w + \lceil log_2(d+1) \rceil + n,\ 4r\sqrt{\varepsilon_x} + \varepsilon_w\right)$-BE, denoted as U_Φ, of the diagonal $K \times K$ matrix encoding the vector corresponding to the output of a KAN layer 1.6:

$$\Phi\left(\frac{1}{N}\sum_{p=1}^{N}\phi_{p1}(x_p), \ldots, \frac{1}{N}\sum_{p=1}^{N}\phi_{pK}(x_p)\right) \tag{2.6}$$

This step is very similar to the previous one, an LCU operation is performed. In this case, the index to iterate is N and hence, the Hadmard gates are applied to the n qubits that encode the input space, which are now set as anzilla qubits in this step. This is performed through the circuit in Figure 2.6.

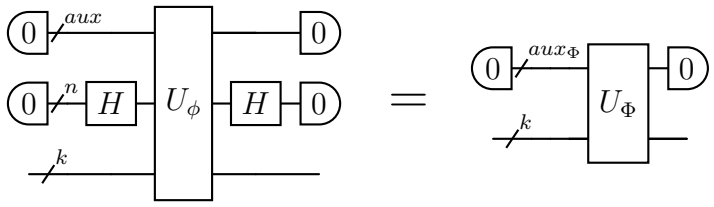


Figure 2.6: Circuit implementing the SUM step for constructing the output BE U_Φ

The unitary U_Φ, is the output of the CHEB-QKAN layer.

Combining layers

To combine layers, one just needs to substitute the input U_x by U_Φ of the previous layer and perform the exact same steps as previously, forming a BE, denoted as U_{CHEB}, of the diagonal matrix corresponding to the output of a KAN:

$$\mathrm{KAN}(\mathbf{x}) = \Phi^{(L)} \circ \ldots \circ \Phi^{(1)}(\mathbf{x}) \tag{2.7}$$

Strengths and caveats of CHEB-QKAN

Authors explain that the main strengths of CHEB-QKAN are the following:

- The number of qubits needed by CHEB-QKAN grows linearly with the number of layers.
- CHEB-QKAN is suitable for quantum input since BEs of the probability amplitudes can be constructed efficiently [49]. This may allow to solve

classically intractable problems such as quantum phase classification [50, 30] or to perform multivariate state preparation, which consists in prepare a quantum state with desired concrete amplitudes [51].

Besides that, CHEB-QKAN presents two powerful caveats:

- The number of BEs, and therefore of quantum gates, needed to construct an L-layer architecture of a CHEB-QKAN, grows exponentially ($\mathcal{O}(d^{2L}/2^L)$) with the number of layers [30]. For that reason, QKAN is limited to shallow architectures, limiting the potential uses of QKAN.

Authors propose that CHEB-QKAN could be a useful tool for regression tasks and state that there is a need of work to empirically compare this model with other quantum architectures.

2.2.2 VQKAN, Adaptive VQKAN and EVQKAN

These 3 versions have been published by the same group of authors [1]. All three architectures are based on the same concept since Adaptive VQKAN and EVQKAN are direct evolutions of VQKAN.

Method

The main idea of these methods [3, 2, 1] consists in performing three steps. First, classical data is transformed according to learnable functions based on B-Splines [38] defined as:

$$\phi(\mathbf{x}) = \sum_{i\in\{0,\dim(\mathbf{x})\}} 2\arccos(E_f(x_i)) + \sum_{s=0}^{N_g-1}\sum_{l=0}^{N_s-1} c_s B_l(x_i) \tag{2.8}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the classical data, $E_f(x_i) = \frac{x_i}{\exp(-x_i)+1}$ is the Fermi-Dirac-like expectation energy-like value [52], c_s are the learnable parameters of these architectures, and form the control points of spline basis functions $B_l(x_i)$ up to degree N_s with grid size N_g.

After that, $\phi(\mathbf{x})$ is encoded into some gate angles of a particular ansatz which depends on the method. Finally a classical output is obtained as the expected value of an observable. The observable is defined as a Hamiltonian [53] which is a linear combination of Pauli matrices.

The difference between all three methods come from the ansatz used in each case:

VQKAN [1] presents two different fixed ansatzes, authors named them *canonical* and *compact* since second one uses less rotation gates. Adaptive VQKAN [2]

does not have a fixed ansatz, in this case, the circuit grows iteratively by adding quantum gates from a predefined pool based on optimization feedback. Finally EVQKAN [3] uses an ansatz based on a tiling technique, authors comment that this tool allows the system to achieve higher accuracies than VQKAN and traditional Quantum Neural Networks (QNNs) [54].

Results

All three architectures have been tested and compared with a traditional QNN for regression and classification problems.

Regression problem inputs 4-dimensional points sampled from $[-1, 1]^4$ and used Function 2.9 to synthetically generate the dataset. This function has range $[1, e^2] \approx [1, 7.39]$, and since it was also used within the experiments of this project, a more detailed description can be found in Section5.1.

$$f_{exp}^{aim}(\mathbf{x}) = \exp\left(\sin(x_0^2 + x_1^2) + \sin(x_2^2 + x_3^2)\right) \tag{2.9}$$

Classification problem inputs 2-dimensional points sampled from $[0, 1]^2$. First dimension of input points is first transformed according to the function:

$$f^{class}(x_0) = \exp(d_0\sqrt{x_0}+d_1)+d_2\sqrt{1 - d_3\sqrt{x_0}^2}+\cos(d_4x_0+d_5)+\sin(d_6\sqrt{x_0}+d_7) \tag{2.10}$$

where $d_k \in [0, 1]$, with k ranging from 0 to 5, represents random coefficients for the various cases. Then, the obtained $f^{class}(x_0)$ are normalized to the $[0, 1]$ domain, generating $f_{norm}^{class}(x_0)$. Finally, classification is performed according to the boundary function:

$$f_{class}^{aim} = \begin{cases} -1 & \text{if } f_{norm}^{class}(x_0) \geq \sqrt{x_1} \\ 1 & \text{if } f_{norm}^{class}(x_0) < \sqrt{x_1} \end{cases} \tag{2.11}$$

As this function was also used to perform the experiments featured in this work, a graphical representation of a dataset generated from this distribution can be visualized in Figure 5.1. Within the authors settings, this classification task, behaves as a regression problem with a final decision boundary.

Each task was repeated 10 times with different initial conditions for all of the tested models, with 50 fixed test points for each task and 10 randomly sampled training points for each run. Both problems used as training loss function a weighted sum of absolute distances, where weights are equal for each method. COBYLA was selected as optimizer. For testing purposes, the Sum of Absolute Distances (SAD) error between predicted and real values was selected:

$$SAD = \sum_{i=1}^{i=50} |f^{aim}(\mathbf{x})_i - \hat{f}^{aim}(\mathbf{x})_i| \tag{2.12}$$

Authors specify that all numerical simulations were performed using blueqat SDK [55] and Intel Core i7-9750H.

Results are summarized in tables 2.2 for regression task and 2.3 for classification problem.

Method	**Average**	**Median**	**Minimum**	**Maximum**
QNN	25.97	25.69	14.54	35.56
VQKAN	22.61	22.96	19.88	24.68
AVQKAN	22.38	21.93	16.90	28.29
EVQKAN	15.09	15.12	13.12	18.64

Table 2.2: Regression sum of absolute distances: average, median, minimum and maximum for 10 runs. (AVQKAN corresponds to Adaptive VQKAN)

Method	**Average**	**Median**	**Minimum**	**Maximum**
QNN	30.16	26.06	18.39	47.95
VQKAN	41.98	42.47	35.76	49.32
AVQKAN	49.73	49.64	48.78	51.61
EVQKAN	29.63	32.45	17.35	39.01

Table 2.3: Classification sum of absolute distances: average, median, minimum and maximum for 10 trials. (AVQKAN corresponds to Adaptive VQKAN)

Chapter 3

OBJECTIVES

As previously mentioned, the state-of-the-art model CHEB-QKAN [30] restricts the basis functions to Chebyshev polynomials. This constraint limits the expressive power of QKAN-based models [56], potentially reducing their ability to approximate functions that are difficult to represent with Chebyshev polynomials, such as those with fractional growth, non-differentiable corners, or sharp peaks (e.g. $\sqrt{x}$) [57, 58].

Moreover, there is currently no validated and publicly available implementation of QKAN-based models, which prevents any experimental analysis. Similarly, the proposed implementations of VQKAN [1], AVQKAN [2], and EVQKAN [3] have not released their source code either, making it impossible to use them for further investigation.

Considering the issues within the current state-of-the-art, this work presents two main objectives:

The first objective is to increase the expressivity of QKAN-based models. To address this, alternative constructions using more general sets of basis functions were explored.

The second objective is to address the lack of publicly available empirical research on QKAN-based models. This will enable a direct comparison between CHEB-QKAN [30] and the alternative QKAN approaches proposed in this work, as well as facilitate a comparison between QKAN-based results and those obtained from VQKAN [1], AVQKAN [2], and EVQKAN [3]. Besides that, a baseline for future research will be established.

Chapter 4

METHODS

This chapter describes the proposed tools to achieve the objectives presented in the previous Chapter 3. For the purposes of this project, all code has been written using the PennyLane framework version 0.40.0 [59] (A brief introduction to PennyLane framework can be found in Appendix A.8), built in Python 3. Throughout the development, ChatGPT [60] was used to support methodical coding tasks and to assist in the writing of this report.

The chapter begins by introducing the two methods proposed to address the first objective of the project, including the rationale behind why they could effectively overcome the identified limitations. It then presents a proposed framework for building classical-to-classical models based on the QKAN architecture, directly tackling the second objective by enabling the implementation, testing, and evaluation of the proposed methods.

4.1 GFCF Transformation

This section presents an approach to address the first objective of this project. It consists in introducing the Generalized Fractional-order of the Chebyshev Functions (GFCF) [61] (A detailed explanation of GFCFs can be found in Appendix A.7) in the first layer of a CHEB-QKAN. The section begins by motivating its inclusion. Next, the implementation details are presented. Finally, a model naming convention is introduced.

4.1.1 Why GFCF?

The main motivation for introducing the GFCF transformation is to improve the expressiveness [56] of classical-to-classical QKAN-based models to improve the

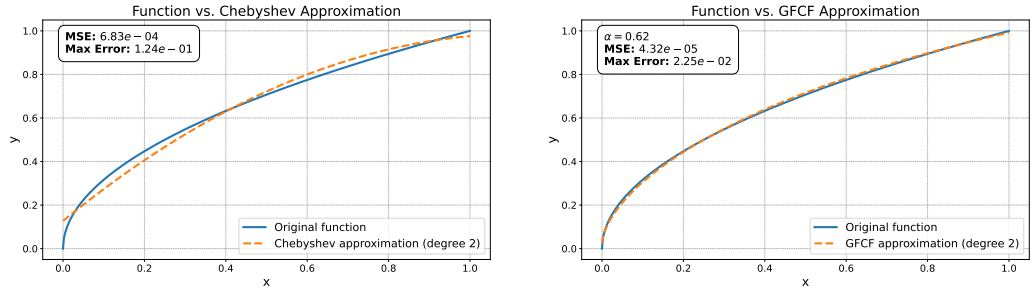
adaptability of the models to nonlinear functions, particularly those that exhibit non-polynomial behavior or non-smooth regions.

In many practical problems, the exact structure of the solution is not known, or it is known to involve fractional powers, as is the case with the Thomas-Fermi equation, whose solution behaves like $t^{1/2}$ [61]. Classical Chebyshev polynomials can struggle in such contexts, especially near domain boundaries or where the function has vanishing derivatives [57, 58]. By applying the following fractional transformation to input data x_{in} prior to the formation of Chebyshev polynomials:

$$x_{gfcf} = 1 - 2\left(\frac{x_{in}}{\eta}\right)^{\alpha} \tag{4.1}$$

where $\alpha, \eta \in \mathbb{R}_+$, GFCFs generate basis functions that are better suited for approximating functions with fractional growth or irregular smoothness, leading to more robust and accurate univariate approximations. GFCFs are denoted as ${}_{\eta}FT_r^{\alpha}(x) = T_r(x_{gfcf})$, where r is the degree of the GFCF.

Figure 4.1 shows the approximation of $f(x) = \sqrt{x}$ under $[0, 1]$ domain by using both, Chebyshev and GFCF basis functions. GFCF method presents lower MSE and lower maximum absolute error. Absolute error refers to the absolute difference between predicted and real points.



(a) Chebyshev approximation.

(b) GFCF approximation.

Figure 4.1: Approximation of $f(x) = \sqrt{x}$ using (a) Chebyshev and (b) GFCF method with $\alpha = 0.62$ (degree 2). MSE: (a) 6.83×10^{-4}, (b) 4.32×10^{-5}. Max. Abs. Error: (a) 1.24×10^{-1}, (b) 2.25×10^{-2}.

For these reasons the GFCF transformation may be a helpful tool to enhance the expressivity of classical-to-classical QKAN-based models.

4.1.2 Outlining GFCF transformation for QKAN context

GFCF transformation is a technique designed for QKAN-based models that receive a classical input and use a QKAN unitary as the main component of their

quantum circuit. In practice, it can be seen as a classical preprocessing step, and therefore can be applied not only to CHEB-QKAN architectures but to any QKAN-based model. To apply it, the N-dimensional input vector must first be normalized to the interval $[0, \eta]^N$. The procedure for this normalization will be described in Section 4.3.3.

Assuming $\mathbf{x}_{\text{norm}} \in [0, \eta]^N$ represents the normalized version of an original input $\mathbf{x}_{in} \in \mathbb{R}^N$, it is transformed according to Equation 4.2:

$$\mathbf{z} = 1 - 2\left(\frac{\mathbf{x}_{\text{norm}}}{\eta}\right)^{\alpha} \tag{4.2}$$

where $\alpha, \eta \in \mathbb{R}_{>0}$.

In this way, a vector $\mathbf{z} \in [-1, 1]^N$ is obtained and the diagonal matrix containing it will be used to create the input BE for the QKAN construction.

The GFCF transformation comes with negligible additional computational cost as it is only performed once prior to data ingestion of the QKAN architecture.

By making the α parameter learnable, the expressive power of the models increases as basis functions of the first layer will adapt themselves to any fractional growth of the data. However, this has a trainability cost, as the parameter space [62] grows.

The parameter η can be effectively omitted in practice, as its influence is absorbed by the output domain of the Hadamard Test used to extract classical values from a QKAN.

4.1.3 GFCF transformation implication over CHEB-QKAN

Particularly for the CHEB-QKAN case, by performing the GFCF transformation on the input classical data, a modification of the CHEB step of the first layer is induced. With this transformation, the CHEB step constructs a diagonal BE of the matrix:

$$\text{diag}\left(\underbrace{{}_{\eta}FT_r^{\alpha}(x_{norm_1}), \ldots, {}_{\eta}FT_r^{\alpha}(x_{norm_1})}_{K}, \ldots, \underbrace{{}_{\eta}FT_r^{\alpha}(x_{norm_N}), \ldots, {}_{\eta}FT_r^{\alpha}(x_{norm_N})}_{K}\right) \tag{4.3}$$

And therefore, GFCFs are successfully introduced as basis functions for the first layer of a CHEB-QKAN model.

4.1.4 GFCF Model Naming

For clarity in model naming, the following convention is adopted: classical-to-classical QKAN models that incorporate the GFCF transformation will be referred

to as GFCF-<QKAN>, where QKAN represents the specific QKAN variant being used (e.g., GFCF-CHEB-QKAN). Models that do not include this transformation will be referred to as Plain-<QKAN>.

4.2 Flexible-QKAN

Flexible QKAN (Flex-QKAN) is the second approach proposed to meet the first objective of this project. This section begins with a brief summary of the contribution and its motivation, continues with an explanation of the implementation and finishes with the introduction of some useful notation.

4.2.1 Flex-QKAN in a nutshell

Flex-QKAN is a novel QKAN architecture that constructs U_{FLEX} as its output BE by parameterizing the QSVT angles to form learnable basis functions. In contrast to CHEB-QKAN, Flex-QKAN is not restricted to a single basis of functions. This flexibility, increases the expressivity of the model, allowing it to discover alternative functions whose linear combinations may converge to a more optimal solution than for the CHEB-QKAN case with the same number of iterations. However, increasing the size of the parameter space comes at a trainability cost, as there are more parameters to optimize.

4.2.2 The FLEX step

Flex-QKAN modifies the CHEB step to achieve its objective. In this case, the name FLEX step is proposed, and the difference with its counterpart is that angles used are not restricted to form Chebyshev basis, instead, they are learnable.

The FLEX step constructs, for each approximation degree $r \in [1, d-1]$, a $(1, a_x + 1, 4r\sqrt{x})$-BE [30], denoted as U_{F_r}, of the diagonal matrix:

$$diag\left(\underbrace{F_r(x_{norm_1}),\ldots,F_r(x_{norm_1})}_{K},\ldots,\underbrace{F_r(x_{norm_N}),\ldots,F_r(x_{norm_N})}_{K}\right), \quad (4.4)$$

where F_r is an arbitrary polynomial function of degree r containing only even or only odd powers (as restricted by the QSVT framework [45]), depending on the parity of r that forms the r-th basis function for layer l.

This construction requires $\mathcal{O}(r)$ applications of the QKAN input U_x.

The ansatz used for each approximation degree r is the same as the CHEB step (see Figure 2.3), however, in this case, the β parameters are not fixed and the unitary implemented is U_{F_r} instead of U_{T_r}.

All the remaining steps of the network construction remain equal to those from CHEB-QKAN.

4.2.3 The U_F Unitary

In the end of its construction, a Flex-QKAN layer will implement the unitary U_F, which is a $(1, a_x+1+a_w+\log_2(d+1)+n, 4d\sqrt{\varepsilon_x}+\varepsilon_w)$ - BE [30] of the diagonal matrix:

$$\text{diag}\left(\frac{1}{N}\sum_{p=1}^{N}\phi_{p1}(x_p),\ldots,\frac{1}{N}\sum_{p=1}^{N}\phi_{pK}(x_p)\right) \tag{4.5}$$

where ϕ_{pq} is the learnable activation function between input dimension p and output dimension q of the layer. It is defined as:

$$\phi_{pq}(x_p)=\frac{1}{d+1}\sum_{r=0}^{d}w_{pq}^{(r)}F_r(x_p)$$

with $w_{pq}^{(r)}\in\mathbb{R}$ being the weight value for the learnable basis function F_r between input dimension p and output dimension q of the layer.

4.2.4 The U_{FLEX} Unitary and its BE Complexity

As for the CHEB-QKAN architecture, all layers can be recursively stacked, by taking U_F as the input BE for the next layer. This forms the U_{FLEX} unitary.

Assuming an L-layer Flex-QKAN architecture with maximum approximation degree d, the construction of U_{FLEX} requires a total of $\mathcal{O}(\frac{d^{2L}}{2^L})$ applications of BEs of the classical data and weight vectors. This is equal to the CHEB-QKAN [30] architecture as Flex-QKAN modifies the parametrization of certain gates but not the number of gates.

4.3 Classical-to-Classical QKAN framework

This section introduces the Classical-to-Classical QKAN framework (CCQKAN). CCQKAN addresses the second objective of this work. It provides an open-source framework [1] designed to implement QKAN-based models that take classical data as input and return a classical output. It integrates both classical and quantum processing stages, thereby constructing the complete pipeline for classical-to-classical modeling.

[1] https://github.com/JavierGonzalezOtero02/CCQKAN

4.3.1 Models generated by CCQKAN

CCQKAN provides the user flexibility in both the classical and quantum processing stages. On the classical side, it supports both Plain and GFCF, (with or without trainable α), processing methods. For the quantum processing step, it allows the use of CHEB-QKAN or Flex-QKAN.

This enables the construction of six different models based on the user choices, these are: Plain-CHEB, Plain-Flex, GFCF-CHEB-0, GFCF-CHEB-1, GFCF-Flex-0 and GFCF-Flex-1, where GFCF models add the suffix 0 or 1 acting as a trainability flag for α. The QKAN suffix is omitted as it adds redundancy. Figure 4.2 outlines all possible combinations of constructing elements.

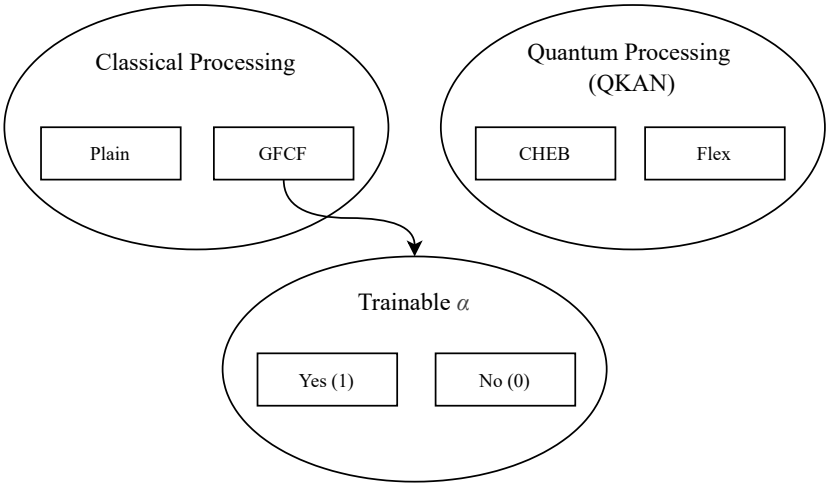


Figure 4.2: Diagram of all possible models that CCQKAN can construct depending on their classical processing method, quantum processing architecture and, in case of being a GFCF model, whether α is trainable or not.

The main construction components of the six models are outlined below.

4.3.2 Models Initialization

CCQKAN is constructed as a Python class, and its main components for initializing a model are outlined below.

Arguments Selection

Depending on the initialization arguments, an instantiated CCQKAN object defines a specific model configuration. The main arguments include:

- **Network structure**: A list of integers defining the QKAN architecture, as shown in 1.3.2, for example: $[4, 2, 1]$.
- **Degree of approximation**: An integer setting the maximum degree of approximations.
- **GFCF flag**: A boolean indicating whether to apply the GFCF transformation to the input data. This parameter allows to distinguish between Plain if it is set to False and GFCF models otherwise.
- α **and** η: Floats greater than 0. α is the value of the GFCF parameter. η is the second parameter of GFCF, used to normalize the input, by default it is set to 1. These two values are only used if GFCF flag is True.
- **Trainable** α **flag**: Only usable if GFCF flag is set to True. It specifies whether the GFCF α should be either optimized during training or kept fixed.
- **Train angles flag**: Determines whether the QSVT angles are trainable, hence selecting between Flex-QKAN (True) and CHEB-QKAN (False) quantum processing block.
- **Input domain**: A list of tuples indicating the domain of each input dimension. If unknown, a wider range is recommended to ensure compatibility.
- **Output domain**: A list indicating the initial target domain. Its values are trainable, but should initially be set to match the minimum and maximum values of the target training data for optimal rescaling.

All these arguments are stored as model attributes so they can be accessed throughout the different steps of models construction.

Table 4.1 shows the combination of key arguments to construct each model type.

Model	GFCF flag	Trainable α flag	Train angles flag
Plain-CHEB	False	-	False
Plain-Flex	False	-	True
GFCF-CHEB-0	True	False	False
GFCF-CHEB-1	True	True	False
GFCF-Flex-0	True	False	True
GFCF-Flex-1	True	True	True

Table 4.1: Argument selection for constructing all possible six models within the CCQKAN framework.

Qubits Management - Wires construction

PennyLane relies on the concept of wires to index qubits of a quantum subsystem (see Appendix A.8).

In CCQKAN, this structure is implemented using two main components stored as model attributes.

First, a dictionary of dictionaries is created. The outer dictionary maps each layer index to a subdictionary that assigns lists of wire indices to specific roles. Figure 4.3 shows an example of this structure for an architecture with layers $[2, 2, 1]$.

```
Layer 0:
  ad: [0, 1], aw: [2, 3], cheb: [4], ax: [5],
  n: [6], k: [7]
Layer 1:
  ad: [8, 9], aw: [10, 11], cheb: [12],
  ax: [0, 1, 2, 3, 5, 6], n: [7], k: [13]
```

Figure 4.3: Wire dictionary for $[2, 2, 1]$ architecture. *ad* are the anzilla qubits used for the LCU step, *aw* are the anzilla qubits for weights BE, *cheb* is the anzilla qubit for QSVT, *ax* are anzilla for input BE, *n* and *k* are qubits encoding input and output respectively.

Second, a sorted list of all wire indices used throughout the architecture is created. It includes an extra qubit reserved for performing the Hadamard test. Figure 4.4 corresponds to the wires list for the Figure 4.4 example.

```
Wires List: [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14]
```

Figure 4.4: Wire list for $[2, 2, 1]$ architecture. Qubit 14 is used for the Hadamard Test.

Parameter Initialization

All parameters of CCQKAN models are stored as attributes constructed as PennyLane numpy arrays enabling its differentiation option if selected (see Appendix A.8). From now on, all PennyLane numpy arrays will be referred to simply as arrays or trainable arrays, differentiating on whether it is non-trainable or trainable.

Four steps are performed to create and initialize parameters:

1. Depending on user's choice, weights of basis functions are initialized to random between -1 and 1 or to 0.5.

2. A two-element trainable array is initialized using the *Output domain* argument values.

3. If GFCF flag is activated, the α parameter is created as an array. It is set to trainable depending on the value of the argument *trainable α flag*.

4. QSVT angles array is initialized to form Chebyshev basis functions, even for the Flex case, as they form a well-known basis of functions, hence acting as a good starting point. This is done through the PennyLane method *qml.poly_to_angles*, taking as argument each of the Chebyshev polynomials in monomial basis. If a Flex-QKAN is being constructed (*Train angles flag* is True), then, angles are set as trainable.

Circuit Construction

The last step for initializing a model through CCQKAN consists in creating its quantum processing circuit based on QKAN. This circuit will be formed by a QKAN block and a Hadamard Test to extract a deterministic real value in $[-1, 1]^{K_{out}}$ with K_{out} being the output dimension of the network.

At its core, this circuit integrates all the QKAN steps, becoming itself, first implementation of a QKAN through the implementation of U_{CHEB} and U_{FLEX}.

The circuit is stored as an attribute and is constructed as a PennyLane *qnode* function (See Appendix A.8).

The input of the *qnode* is a preprocessed diagonal matrix with values in $[-1, 1]^N$. The first operation performed by the circuit is the simulation of the BE of this matrix, which are then used as input for the selected QKAN pipeline. This step is implemented using the *qml.BlockEncode* function provided by PennyLane.

Once the input has been block-encoded, all the steps of QKAN [30] are executed to construct the unitaries U_{CHEB} or U_{FLEX} depending on the model constructed. The second step differs depending on the model type: the CHEB step is used to construct a CHEB-QKAN, and the FLEX step is used in the Flex-QKAN case.

To extract a classical output, the constructed unitary (U_{CHEB} or U_{FLEX}) is used as the principal block of a Hadamard Test, forming the two circuits in Figure 4.5.

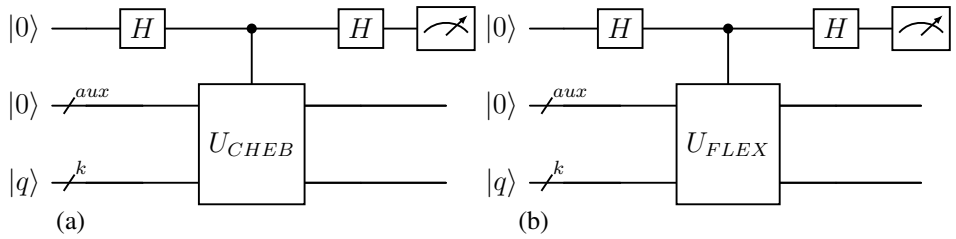


Figure 4.5: Quantum circuits for extracting classical outputs through a Hadamard Test from: (a) CHEB-QKAN, (b) Flex-QKAN where aux are all anzilla qubits used in the QKAN unitary construction, k are the qubits representing the output

4.3.3 End-to-End Forward Pipeline

Models created through the CCQKAN framework receive a classical input $\mathbf{x}_{\text{in}} \in \mathbb{R}^{N_{in}}$, corresponding to a dataset sample, and output a classical vector $\mathbf{x}_{\text{out}} \in \mathbb{R}^{K_{out}}$. The full forward process from input ingestion to output generation for the classical sample $\mathbf{x}_{in}$ is outlined below.

Classical Preprocessing

The first step inputs the classical $\mathbf{x}_{in}$ vector. Its objective is to prepare the diagonal matrix that will be used as input for the model's quantum circuit.

The implementation of this step depends on the type of classical processing type selected, either Plain or GFCF.

Plain models For the Plain models, input is scaled to $[-1, 1]^N$ domain through a min-max scaling. This is done by applying the following transformation:

$$\mathbf{x}_{plain} = 2\frac{\mathbf{x}_{in} - min_{in}}{max_{in} - min_{in}} - 1 \tag{4.6}$$

Where min_{in} and max_{in} are real numbers representing the scaling bounds, and are set for each dimension through the *Input domain* argument within model construction. Then, $\mathbf{x}_{plain}$ is then used to create the circuit's input diagonal matrix as $\mathbf{X}_{plain} = diag(\mathbf{x}_{plain})$.

GFCF models GFCF models scale their input to $[0, \eta]^{N_{in}}$ domain to fulfill GFCF input requirements [61]. This is done applying the following transformation:

$$\mathbf{x}_{norm} = \eta\frac{\mathbf{x}_{in} - min_{in}}{max_{in} - min_{in}} \tag{4.7}$$

Then, GFCF transformation in Equation 4.2 is applied to create $\mathbf{x}_{gfcf}$, which is used to create $\mathbf{X}_{gfcf} = diag(\mathbf{x}_{gfcf})$ as the circuit input diagonal matrix.

Quantum Processing

This step receives the previously created diagonal matrix. Its objective is to obtain a vector $\mathbf{x}_h \in [-1, 1]^{K_{out}}$ output corresponding to the solution model of a QKAN.

To achieve this objective, the diagonal matrix is used as input to the model's quantum circuit.

The specific circuit used to process this input depends on the chosen quantum processing option. In all cases, the circuit consists of a Hadamard test, with the difference being that the main block of the Hadamard test is U_{CHEB} for CHEB-QKAN and U_{FLEX} for Flex-QKAN.

To extract the output $\mathbf{x}_h$, the circuit must be executed K_{out} times, one for each output dimension, (Execution here refers to a quantum test defined in 1.1.2[16]).

To measure the q-th output dimension, where $q \in [0, K_{out} - 1]$, the input state $|\psi\rangle = |0\rangle \otimes |0\rangle_a \otimes |q\rangle_k$ must be applied to the circuit (See Figure 4.5). Here, a refers to the anzilla qubits used by QKAN, and $k = \lceil \log_2(K_{out}) \rceil$ corresponds to the output qubits of the QKAN. This step projects the q-th output dimension of the unitary operator U_{CHEB} or U_{FLEX} depending on the case, hence, obtaining the corresponding classical value. Once this step has been performed for every q-th dimension, the $\mathbf{x}_h \in [-1, 1]^{K_{out}}$ can be constructed by simply forming an array.

Classical Post-Processing

This last step receives the previously created $\mathbf{x}_h \in [-1, 1]^{K_{out}}$ as input. Its objective is to construct the forward's output $\mathbf{x}_{out}$ by rescaling $\mathbf{x}_h$ to a proper output domain. To this end, the inverse preprocessing scaling operation is performed. Therefore, its implementation depends on the model constructed. Besides that, output domain has been set learnable but it is initialized through the *output domain* argument.

Plain models rescale their output according to the following mapping:

$$\mathbf{x}_{out} = (max_{out} - min_{out}) \frac{\mathbf{x}_h + 1}{2} + min_{out} \tag{4.8}$$

Where max_{out} and min_{out} form the trainable output domain. GFCF models perform the following rescaling:

$$\mathbf{x}_{out} = \mathbf{x}_h \frac{max_{out} - min_{out}}{\eta} + min_{out} \tag{4.9}$$

Finally, $\mathbf{x}_{out}$ is returned.

Chapter 5

RESULTS

This chapter presents the main results obtained from the six proposed models. It begins with a detailed explanation of the experimental setup used to evaluate their feasibility, continues presenting the obtained experimental results, and concludes with a discussion of these results.

5.1 Experimental Setup

A total of three different tasks were conducted: Classification task, Exponential Regression and Polynomial Regression. The first two tasks are aimed to compare QKAN architectures with baseline VQKAN [1], AVQKAN [2] and EVQKAN [3] results.

Training was implemented leveraging PennyLane Numpy interface [59], which integrates autograd to automatically differentiate operations containing PennyLane Numpy arrays and operations. PennyLane Qnodes can automatically be differentiated through this feature, allowing gradient-based optimization methods. All models were trained using CSUC Pirineus III HPC resources [63].

5.1.1 Tasks Description

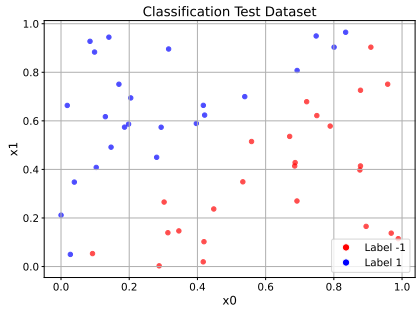
A total of 10 independent runs were conducted for each model on every task. For each run, a new model instance was initialized with random weights, and a distinct training dataset was used. Each task had a fixed test dataset, making inter-model comparisons possible.

Classification task

Datasets: Features were composed of randomly sampled points in $[0, 1]^2$. Their target values were computed according to Equation 2.10, normalized to the $[0, 1]$

range, and assigned a label of -1 or 1 based on Equation 2.11. The selected domain matches that of the baseline models to ensure a fair comparison, as the evaluation metric used is sensitive to the spatial characteristics of the data.

Training datasets were composed of 45 samples whereas the test dataset was formed by 50 samples just like in baseline as the evaluation metric is also sensitive to the size of the dataset. Test dataset can be visualized in Figure 5.1.



(a)

x_0	x_1	$target$	$label$
0.09	0.05	0.30	-1
0.19	0.57	0.43	1
0.35	0.15	0.59	-1
0.40	0.59	0.63	1
0.54	0.70	0.74	1

(b)

Figure 5.1: Test dataset used for the classification task. (a) Visualization of the test distribution generated from Equations 2.10 and 2.11. (b) Sample of test data points including target and label, where $target = f_{norm}(x_0)$.

Training procedure: Following the baseline implementation, classification models were trained as 1-dimensional regression models. The optimization procedure was carried out classically with the following hyperparameters:

- Optimizer: PennyLane Adam Optimizer with $StepSize = 0.3$, $\beta_1 = 0.9$, $\beta_2 = 0.99$, and $\varepsilon = 1\text{e}{-08}$. The parameters β_1 and β_2 control the update of the first and second momentum estimates, respectively, and ε corresponds to an offset added for numerical stability.
- Number of epochs: 20
- Output domain: $[min_{train\_y}, max_{train\_y}]$ where min_{train_y} and max_{train_y} form the training target domain.
- QKAN architectures: $[1, 2, 1]$ with maximum approximation degree $d = 2$ was used for all six models.
- Training loss (objective function): Mean Squared Error (MSE) between predicted and real test target values was used:

$$MSE = \frac{1}{50} \sum_{i=1}^{i=50} (f^{class}(x_0)_i - \mathbf{x}_{out_i})^2 \tag{5.1}$$

where $f^{class}(x_0)_i$ is the target function defined in Equation 2.10

One *std* node with 64gb of RAM was used for training.

Evaluation metrics: Metric used to compare with baseline results was SAD error between predicted and target test values. This metric is defined in 2.12. Additionally, Accuracy, Precision and Recall scores were calculated.

Exponential Regression

Datasets: Features, were randomly sampled from the $[-1, 1]^4$ domain as in baseline experiments, allowing a proper comparison. Target was calculated from baseline Equation 2.9, and its range, considering the input domain, is $[1, e^2] \approx [1, 7.39]$. In this task, training datasets were formed by 100 samples. As in baseline, test dataset contained 50 samples. As this dataset inputs 4-dimensional points, a visualization cannot be made, however, Table 5.1 shows some of the test samples used.

x_0	x_1	x_2	x_3	$target$
0.10	0.14	0.36	-0.70	1.84
0.43	-0.12	-0.46	0.74	2.42
0.21	0.98	0.47	-0.68	4.33
0.09	-0.80	0.92	0.23	4.00
-0.15	-0.58	-0.50	-0.75	2.96

Table 5.1: Sample of test dataset used within Exponential Regression task. Generated test samples $target$ is bounded to $[1.08, 7.18]$

Training procedure: Training was performed with the following hyperparameters:

- Optimizer: PennyLane Adam Optimizer with $StepSize = 0.3$, $\beta_1 = 0.9$, $\beta_2 = 0.99$.
- Number of epochs: 55
- Output domain: $[min_{train\_y}, max_{train\_y}]$ where min_{train_y} and max_{train_y} form the training target domain.
- QKAN architectures: $[4, 1]$ with maximum approximation degree $d = 2$ was used for all six models.

- Training loss: MSE between predicted and real test target values was used. This error function is defined in 5.1

One *std* node with 32gb of RAM was used for training.

Evaluation metrics: As in the previous case, the metric used to compare with baseline results was SAD (defined in 2.12) between predicted and target test values.

Polynomial Regression

This task is proposed in this work to allow for a visual evaluation of the models' prediction capacity.

Datasets: In this last experiment, features were composed of randomly sampled points in $[0, 1]^2$, and target values were calculated according to Equation 5.2

$$f^{aim}(\mathbf{x}) = x_0^2 + x_1^2 \tag{5.2}$$

For this task, training datasets consisted of 100 samples and test dataset was formed by 50 samples. Test dataset can be visualized in Figure 5.2.

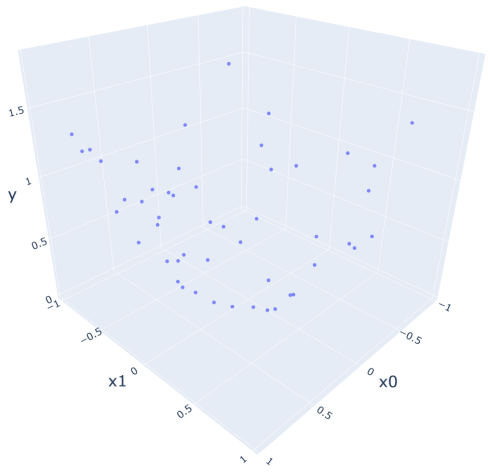


Figure 5.2: Polynomial test dataset, generated from Equation 5.2. To contextualize error metrics, $y \in [0.29, 1.82]$ where $y = f^{aim}$.

Training procedure: In this case, training uses the following hyperparameters:

- Optimizer: PennyLane Adam Optimizer with $StepSize = 0.3$, $\beta_1 = 0.9$, $\beta_2 = 0.99$.
- Number of epochs: 100
- Output domain: $[min_{train\_y}, max_{train\_y}]$ where min_{train_y} and max_{train_y} form the training target domain.
- QKAN architectures: $[2, 1]$ with maximum approximation degree $d = 2$ was used for all six models.
- Training loss objective function: MSE (defined in 5.1) between predicted and real test target values was used.

One *std* node with 32gb of RAM was used for training.

Evaluation metrics: Polynomial Regression used both SAD and the MSE for every attempt.

5.2 Experimental Results

This section shows the obtained evaluation results for every model in all tasks. In addition, an analysis of parameters consumption for every model is assessed.

5.2.1 Classification task

Figures 5.3 and 5.4 present the main results of the classification experiments. One key finding is that GFCF models outperform their Plain counterparts in accuracy, precision, and recall, given a fixed QKAN architecture. This can be observed by comparing the distributions between Plain-CHEB [30] and GFCF-CHEB-0, as well as between Plain-Flex and GFCF-Flex-0 in Figure 5.3. Additionally, Flex variants show consistent improvements over their CHEB equivalents. This is seen by comparing Plain-CHEB [30] to Plain-Flex, and GFCF-CHEB-0 to GFCF-Flex-0. Another relevant observation is that making the α parameter trainable in GFCF models improves their performance, as demonstrated by the difference between GFCF-CHEB-0 and GFCF-CHEB-1 in the same figure.

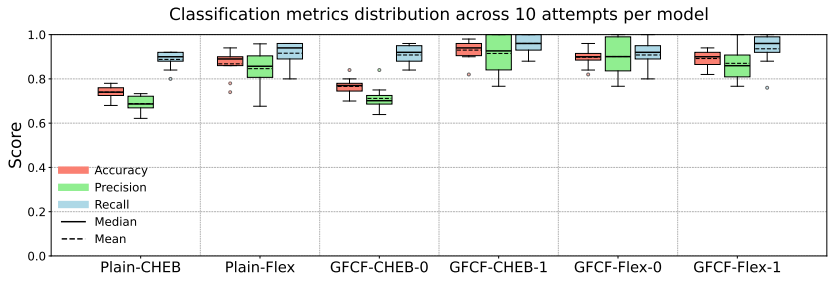


Figure 5.3: Evaluation accuracy, precision, and recall distribution across runs for tested models.

Figure 5.4 shows that QKAN models substantially improved the results of VQKAN [1], AVQKAN [2] and EVQKAN [3] (See Table 2.3) in terms of SAD error.

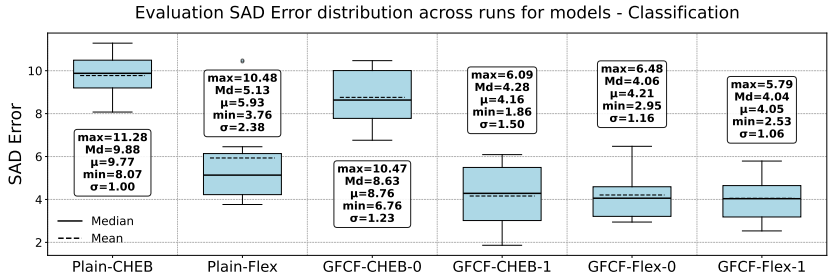


Figure 5.4: Classification evaluation SAD error distribution for tested models. Average (μ), Median (Md), Maximum (max) and Minimum (Min) are presented to compare with VQKAN [1], AVQKAN [2] and EVQKAN [3].

Training time per run was also measured, and its distribution is shown in Figure 5.5. GFCF models with fixed α not only improve performance advantages but also achieve shorter training times than their Plain counterparts. Detailed numerical results can be found in Table D.2.

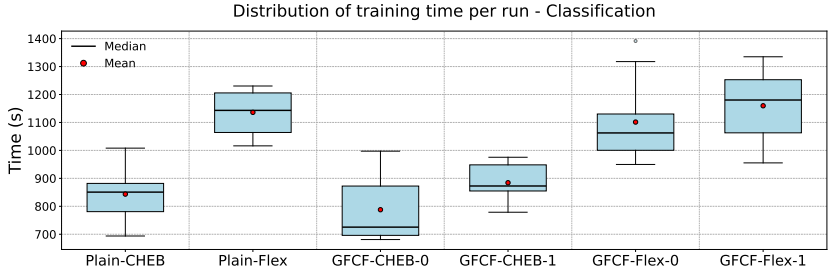


Figure 5.5: Distribution of training time (in seconds) per run for tested models in Classification task.

5.2.2 Exponential Regression

Figure 5.6 presents the SAD error distribution across runs for each model, while Figure 5.7 displays the training time distribution per run.

The same conclusions observed in the classification task hold for the Exponential Regression experiment. And, in this case, Flex-based models consistently outperformed their CHEB counterparts. Moreover, GFCF-Flex-1 surpasses all baseline results [1, 2, 3] (See 2.2).

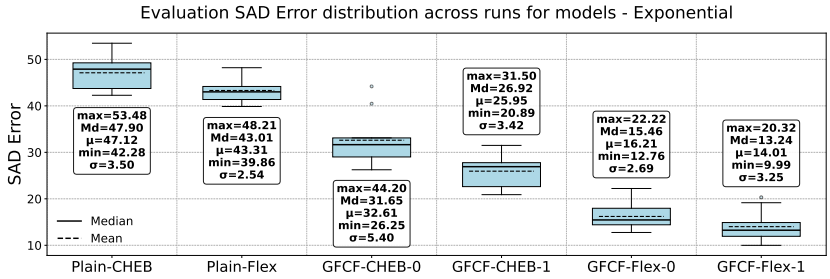


Figure 5.6: Exponential Regression evaluation SAD error distribution across runs for tested models.

The training time distribution, shown in Figure 5.7, reinforces previous findings. GFCF models with non-trainable α achieved faster training compared to their Plain counterparts.

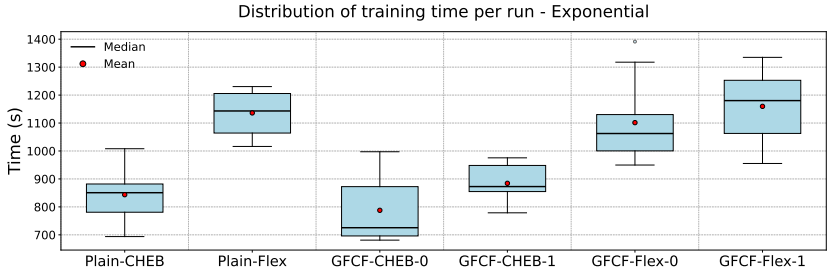


Figure 5.7: Distribution of training time (in seconds) per run for tested models in Exponential Regression task. Numerical results can be found in Table D.3.

5.2.3 Polynomial Regression

As previously stated, the Polynomial Regression task is proposed in this project.

This task revealed a key finding. Flex models superiority compared to their CHEB counterpart was surprisingly higher than for the Classification and Exponential Regression tasks. The same result occurs when α is made trainable, sub-

stantially increasing the performance gap shown in previous tasks. These results are shown in Figure 5.8, which presents the SAD error distribution across models.

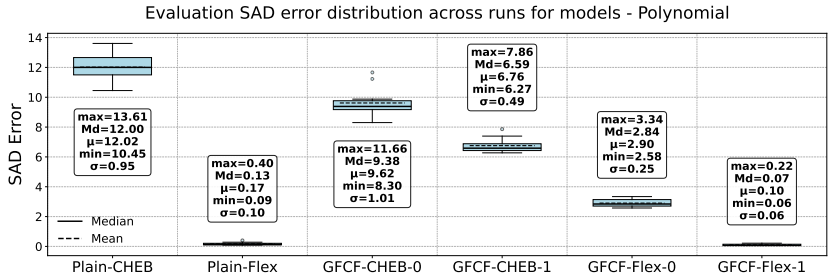


Figure 5.8: Polynomial Regression evaluation SAD error distribution across runs for tested models.

Figure 5.9 visually exposes the superior fitting capacity of GFCF-Flex-1 compared to Plain-CHEB [30] by showing a comparison of predicted and true values for the test dataset in both cases.

(a) Plain-CHEB [30]

(b) GFCF-Flex-1

Figure 5.9: Comparison between true and predicted values of the test dataset for two representative models.

Finally, in this case, training times were almost equal for Plain-CHEB [30] and GFCF-CHEB-0, however Flex models showed higher training times than CHEB models.

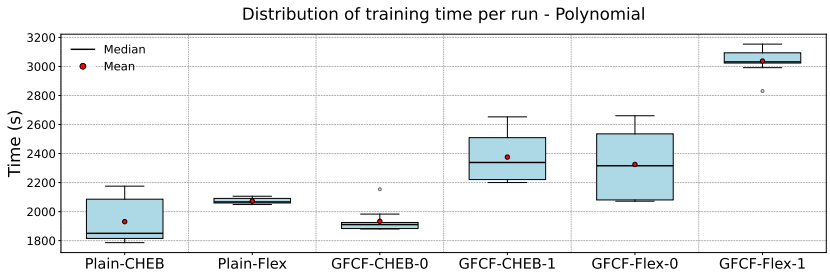


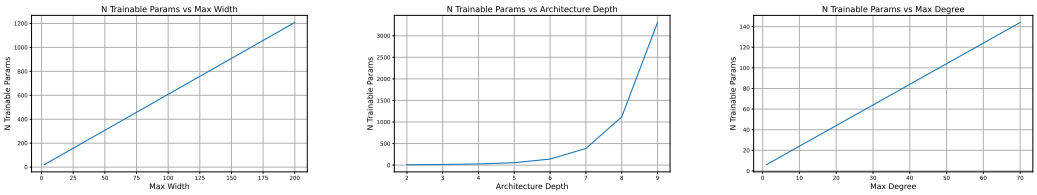
Figure 5.10: Distribution of training time (in seconds) per run for tested models in Polynomial Regression task. Expanded numerical distribution values can be seen in D.4.

5.2.4 Parameter consumption

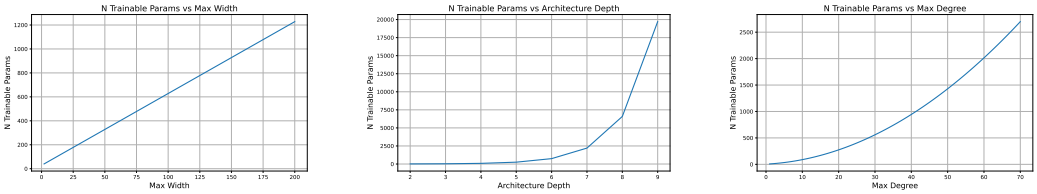
To compare model families, it is useful to analyze their parameter consumption [27]. In this case, the pre- and post-processing stages add only 3 extra parameters at maximum, which are α (GFCF only), min_{out} and max_{out}. Therefore, an analysis has been conducted to examine the parameter consumption as a function of the underlying QKAN architecture.

Figure 5.11 shows how parameter consumption varies for CHEB and Flex models when increasing: the maximum input width for a $[N, 1]$ architecture; the architecture depth for a $[2, \ldots, 2]$ architecture; and the maximum approximation degree d of a $[2, 1]$ architecture.

Both CHEB and Flex models show linear growth in the number of parameters when increasing the input width for a fixed 1-layer architecture $[N, 1]$. When scaling in terms of architecture depth, parameter growth appears exponential, and much faster for Flex models. Finally, increasing the maximum approximation degree d presents a $\mathcal{O}(d^2)$ growth in parameters quantity for Flex models. This behavior is explained as follows: on one hand, QSVT introduces a total of $\mathcal{O}(i)$ angles per i-degree polynomial, with $i \in [0, d]$, and there are $d + 1$ polynomials being linearly combined, hence the total number of extra parameters is $\sum_{i=0}^{i=d-1} d - i + 1$ leading to an angle complexity of $\mathcal{O}(d^2)$. On the other hand, the number of classical weights for a fixed $[2, 1]$ architecture scales as $2d$, that is, $\mathcal{O}(d)$. Hence, the overall parameter complexity becomes $\mathcal{O}(d^2) + \mathcal{O}(d) = \mathcal{O}(d^2)$. In contrast, CHEB models do not add any parameters within the QSVT construction, so the total number of parameters for the same architecture and varying d remains $2d$, i.e., $\mathcal{O}(d)$.



(a) Max. width - CHEB (b) Depth - CHEB (c) Max. degree - CHEB



(d) Max. width - Flex (e) Depth - Flex (f) Max. degree - Flex

Figure 5.11: Total parameters consumption for (First row) CHEB and (Second row) Flex models. (a & d) Expands in maximum input dimension width, fixing a 1-Layer architecture [N, 1]. (b & e) Increases architecture depth fixing 2-Dimensional layers. (c & f) Augments the maximum approximation degree d fixing a $[2, 1]$ architecture.

5.3 Discussion

The results presented in Section 5.2 demonstrate that the methods proposed in this work (GFCF and Flex-QKAN) provide clear improvements over CHEB-QKAN [30].

5.3.1 Impact of GFCF on Adaptability and Efficiency

The motivation under the GFCF transformation was to enhance the expressivity of classical-to-classical QKAN-based models without affecting trainability. This was done by modifying the basis functions used within the first layer to cover non-polynomial constructions.

Results have been promising for both fixed and learnable α cases, obtaining lower SAD errors than Plain models for every experiment performed. In particular, the first case does not present any trainability cost as it adds $\mathcal{O}(1)$ operations to the Plain counterpart, and the second case just adds one extra parameter.

Expressivity was successfully increased for the learnable α case, as contrary to Plain models, the basis functions, (which are orthogonal and hence Universal for any α), are able to learn the fractional properties of data, and hence, adapting better to it. However, despite results are promising for the fixed α case, a deeper

theoretical analysis is needed to explore its expressivity power.

5.3.2 Balancing Expressivity and Trainability in Flex-QKAN and CHEB-QKAN

Similarly to the GFCF transformation, Flex-QKAN was designed to improve the expressivity of CHEB-QKAN and hence, increasing its adaptability. This was done by making the basis functions learnable through the parametrization of QSVT, consequently, allowing the model to select the best basis functions depending on the data characteristics. This allows to capture patterns that would otherwise be difficult to learn for a CHEB-QKAN with a limited number of training iterations.

The results show that Flex-QKAN significantly outperformed CHEB-QKAN [30], particularly in the Polynomial Regression task, where the improvement is clearly visible in Figures 5.6 and 5.9.

However, expanding the parameter space introduces a computational cost. As shown in Figures 5.5 ,5.7 and 5.10, training time of Flex models increases with respect to CHEB ones.

To decide between using CHEB-QKAN or Flex-QKAN, one must consider the trade-off between expressiveness and trainability: if greater expressiveness is required, Flex-QKAN is preferable; if easier trainability is a priority, CHEB-QKAN is the better choice.

Considering that the number of parameters introduced by Flex-QKAN grows as $\mathcal{O}(d^2)$ compared to CHEB-QKAN, this increase typically does not represent a significant memory requirement in most practical scenarios. Therefore, the only situations in which one might prefer CHEB-QKAN over Flex-QKAN are those where the maximum approximation degree d is large enough to make storing the additional $\mathcal{O}(d^2)$ parameters infeasible, or when training and deployment time constraints demand a faster and more light-weight model.

5.3.3 Comparison with VQKAN, AVQKAN, and EVQKAN

In addition to comparisons among QKAN-based models, a comparison with baseline results reported by Hikaru et al. in VQKAN [1], AVQKAN [2], and EVQKAN [3] has been conducted.

The classification task demonstrates that all CCQKAN models significantly outperform their variational counterparts. This can be observed by comparing Figure 5.4 with Table 2.3, where the minimum difference in average SAD error is nearly 10 units. However, such comparisons must be interpreted with caution, as the authors of the baseline counterparts provide very limited information regard-

ing the datasets used and do not report architectural details. For instance, they may have used a significantly smaller number of qubits, which is not specified.

Similarly, the exponential regression task shows that the most expressive model proposed in this work, GFCF-Flex-1, is capable of outperforming all results reported by Hikaru et al. Again, this observation should be treated cautiously due to the same lack of transparency about the architectures used in their experiments.

In conclusion, this comparison should be seen only as an initial reference for future evaluations with different models. It does not support any definitive claims about the superiority of QKAN over VQKAN [1], AVQKAN [2], or EVQKAN [3].

5.3.4 Near-term Perspective for CCQKAN Models

The experimental results have been promising, opening a new line of research focused on maximizing the expressivity of QML models.

Furthermore, the introduction of GFCF suggests potential uses beyond QKAN itself, such as a general-purpose data preprocessing technique for both quantum and classical models.

CCQKAN models inherit the advantages of QKAN models. First, their quantum circuit is efficient in terms of gate complexity with respect to the input dimension N_{in} assuming a $\mathcal{O}(log(N_{in}))$ BE construction of the input. The reason behind this is that as stated in 2.2.1, the gate complexity of an L-layer QKAN with maximum degree d scales as $\mathcal{O}(d^{2L}/2^L)$, and therefore, by factoring with the previous expression, increasing the maximum width of the model implies an $\mathcal{O}(log(N_{in}))$ growth in the number of gates.

Second, the CCQKAN framework allows for a straightforward extension to quantum-to-quantum models. This can be achieved by modifying the internal structure so that, instead of measuring $Re\left(\langle\psi|_q \, U_{\text{QKAN}} \, |\psi\rangle_q\right)$ for every output dimension $q \in [0, K_{out}]$, the resulting unitary U_{QKAN} is used directly as a state preparation unitary for further quantum processing.

It is important to note that this study was conducted under ideal conditions, without noise, quantum decoherence, or hardware constraints. Additionally, it assumes the existence of an efficient implementation of BEs. Further research is needed to explore how such implementations could be realized in practice and to evaluate the viability of CCQKAN in realistic quantum computing environments.

Chapter 6

CONCLUSION

This work contributes to the development of Quantum Kolmogorov-Arnold Networks (QKAN) in two main directions:

First, two methods were proposed to increase the expressivity power of QKAN-based models, and hence tackling the first objective of the project:

- The first method, designed for QKAN networks with classical input, applies the transformation proposed by Kourosh Parand et al. in [61] to the input before forming the initial BE. This technique enables the construction of GFCFs [61] with learnable or fixed α as basis functions in the first layer of a CHEB-QKAN. The empirical results show that SAD errors were consistently reduced for both the fixed and learnable α case, compared to Plain models. GFCFs with learnable α increase the expressivity power of QKAN-based models, as the rational factors modeling the data can be all covered. However, making α learnable comes with a trainability reduction. Contrary, GFCF models with fixed α do not present any reduction in their trainability capacity. However, a deeper theoretical analysis of this technique case needs to be performed to further evaluate their expressive power.

- The second method introduces Flex-QKAN, a new QKAN-based architecture that learns the set of basis functions to be used. This approach increases the expressivity of QKAN models compared to CHEB-QKAN, allowing them to discover better solutions than those limited to Chebyshev polynomials. This flexibility is achieved by making the angles used in the QSVT routine trainable, which results in a quadratic increase in the number of parameters with respect to the maximum approximation degree, and hence in a reduction of the trainability capacity. The results are promising and show substantial improvements over CHEB models across all tasks.

  By combining this method with the GFCF technique, the expressiveness of

the generated model increases even more, as fractional growths can also be better covered by the basis functions of the first layer.

Second, the CCQKAN framework has been proposed as a complete, documented, and open-source implementation of classical-to-classical models that employ QKAN as their primary quantum processing unit. This framework enabled the empirical evaluation of the contributions of this work, including direct comparisons with state-of-the-art CHEB-QKAN, as well as a detailed study of QKAN models on the tasks explored by Hikaru et al. in [1, 2, 3]. Thus, the second objective of this work has also been successfully addressed.

However, there remains significant room for further exploration in the domain of QKAN-based models, motivating a range of directions for future work.

The contributions of this project have been submited to the international conference Quantum Techniques in Machine Learning (QTML).

6.1 Future Work

One potential direction for future research is to increase the flexibility of the activation functions used within QKAN models to enhance their expressive power. This could be pursued by implementing QKAN with activation functions constructed from linear combinations of B-splines [38], which can be approximated with the QSVT framework as discussed in [30]. Unlike Chebyshev polynomials, B-splines are locally adaptive [38], meaning that changes to their structure affect only limited regions. This property allows for fine-grained local adaptation to target functions, potentially improving the model's expressiveness.

A second direction involves improving the architectural efficiency of QKAN. Specifically, a more compact construction could be achieved by directly forming the activation functions within the LCU step using learnable weights, rather than performing the process separately through a multiplication of BEs and the summation.

Third, in this work, the GFCF transformation has only been applied to the first layer of a QKAN. The direct next step is to implement it for the rest of layers. For that, one would have to solve the following problem: *Given a BE U_x that encodes a diagonal matrix X of a vector $x \in \mathbb{R}^N$, find a BE U_z of the diagonal matrix Z such that the diagonal values of Z are the diagonal elements of X transformed by $z = 1 - 2(\frac{x}{\eta})^\alpha$.* This is possible, as an arbitrary power of a unitary can be approximated as demonstrated in [64], and the $1 - 2a$ part can be easily implemented through an LCU step. This would probably improve the results obtained by the GFCF strategy, however, the implications on trainability should be studied. Of course, in this settings, one could use an array of learnable $\alpha's$ to increase even

more the expressiveness of the network by capturing different fractional growth powers.

Fourth, the effectiveness of the GFCF technique has primarily been demonstrated empirically. A more rigorous theoretical analysis should be conducted to better understand its mathematical properties and potential limitations.

Fifth, the CCQKAN framework simulates efficient BE constructions. Further work is needed to implement a hardware-compatible version. A first approach could be to implement the BEs through the FABLE routine [65], which approximates the BE to a certain precision.

Finally, while the models generated with the CCQKAN framework have achieved promising results, these were obtained under controlled, ideal conditions. The next step should involve testing these models on real-world datasets or in noisier environments to evaluate their practical utility and robustness.

Bibliography

[1] Hikaru Wakaura and Andriyan B. Suksmono. The quantum version of kolmogorov-arnold network (kan): Variational quantum kolmogorov-arnold network, June 2024. Preprint, Version 1, available at Research Square.

[2] Hikaru Wakaura, Rahmat Mulyawan, and Andriyan B. Suksmono. Adaptive variational quantum kolmogorov-arnold network, 2025.

[3] Hikaru Wakaura, Rahmat Mulyawan, and Andriyan B. Suksmono. Enhanced variational quantum kolmogorov-arnold network, 2025.

[4] Pradosh K. Roy. Quantum logic gates. `https://www.researchgate.net/publication/343833536_Quantum_Logic_Gates`, 2020. Preprint, accessed via ResearchGate.

[5] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[6] Eleanor G. Rieffel and Wolfgang Polak. An introduction to quantum computing for non-physicists, 2000.

[7] Madhura Rath and Himanshu Date. Quantum data encoding: a comparative analysis of classical-to-quantum mapping techniques and their impact on machine learning accuracy. *EPJ Quantum Technology*, 11(1):72, 2024.

[8] Giacomo Nannicini. An introduction to quantum computing, without the physics, 2020.

[9] Max Born. Quantenmechanik der stoßvorgänge. *Zeitschrift für Physik*, 38(11):803–827, 1926.

[10] Noson S. Yanofsky and Mirco A. Mannucci. *Quantum Computing for Computer Scientists*. Cambridge University Press, 2008.

[11] Michele Mosca. *Quantum Algorithms*. University of Waterloo, 2008. Available online: https://files.batistalab.com/teaching/attachments/chem584/Mosca.pdf.

[12] Ashoke Das and Sahin Islam. Importance of hilbert space in quantum mechanics. *International Journal of Mathematics Trends and Technology*, Volume-66:92–102, 08 2021.

[13] Mathias Soeken, D. Michael Miller, and Rolf Drechsler. Quantum circuits employing roots of the pauli matrices. *Physical Review A*, 88(4), October 2013.

[14] Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A. Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52(5):3457–3467, November 1995.

[15] Kyrylo Simonov, Marcello Caleffi, Jessica Illiano, Jacquiline Romero, and Angela Sara Cacciapuoti. Universal quantum computation via superposed orders of single-qubit gates, 2024.

[16] Jozef Gruska. *Quantum Computing*. McGraw-Hill, London, 1999. Available online: https://www.fi.muni.cz/usr/gruska/qbook1.pdf.

[17] Wafa Elmannai, Varun Pande, Ajay Shrestha, and Khaled Elleithy. Quantum observable. 06 2013.

[18] H. M. Wiseman and G. J. Milburn. *Quantum Measurement and Control*. Cambridge University Press, 2009.

[19] Edgard Muñoz-Coreas and Himanshu Thapliyal. Everything you always wanted to know about quantum circuits, August 2022.

[20] Phillip Kaye, Raymond Laflamme, and Michele Mosca. *An Introduction to Quantum Computing*. Oxford University Press, Oxford, 2007.

[21] Noson S. Yanofsky. An introduction to quantum computing, 2007.

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

[23] Xiaoyu Guo, Takahiro Muta, and Jianjun Zhao. Quantum circuit ansatz: Patterns of abstraction and reuse of quantum algorithm design, 2024.

[24] Nimish Mishra, Manik Kapil, Hemant Rakesh, Amit Anand, Nilima Mishra, Aakash Warke, Soumya Sarkar, Sanchayan Dutta, Sabhyata Gupta, Aditya Dash, Rakshit Gharat, Yagnik Chatterjee, Shuvarati Roy, Shivam Raj, Valay Jain, Shreeram Bagaria, Smit Chaudhary, Vishwanath Singh, Rituparna Maji, and Prasanta Panigrahi. *Quantum Machine Learning: A Review and Current Status*, pages 101–145. 01 2021.

[25] M. Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C. Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R. McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, and Patrick J. Coles. Variational quantum algorithms. *Nature Reviews Physics*, 3(9):625–644, August 2021.

[26] Hongkang Ni, Haoya Li, and Lexing Ying. On low-depth algorithms for quantum phase estimation. *Quantum*, 7:1165, November 2023.

[27] Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y. Hou, and Max Tegmark. Kan: Kolmogorov-arnold networks, 2025.

[28] George Cybenko. Approximation by superpositions of a sigmoidal function. math cont sig syst (mcss) 2:303-314. *Mathematics of Control, Signals, and Systems*, 2:303–314, 12 1989.

[29] Johannes Schmidt-Hieber. The kolmogorov-arnold representation theorem revisited. *CoRR*, abs/2007.15884, 2020.

[30] Petr Ivashkov, Po-Wei Huang, Kelvin Koor, Lirandë Pira, and Patrick Rebentrost. Qkan: Quantum kolmogorov-arnold networks, 2024.

[31] Ammar Daskin. Quantum Kolmogorov-Arnold networks by combining quantum signal processing circuits. 10 2024.

[32] A. Polar and M. Poluektov. A deep machine learning algorithm for construction of the kolmogorov–arnold representation. *Engineering Applications of Artificial Intelligence*, 99:104137, 2021.

[33] Heinrich van Deventer, Pieter Janse van Rensburg, and Anna Bosman. Kasam: Spline additive models for function approximation, 2022.

[34] Karthik Mohan, Hanxiao Wang, and Xiatian Zhu. Kans for computer vision: An experimental study, 2024.

[35] Cristian J. Vaca-Rubio, Luis Blanco, Roberto Pereira, and Màrius Caus. Kolmogorov-arnold networks (kans) for time series analysis, 2024.

[36] Kunpeng Xu, Lifei Chen, and Shengrui Wang. Kolmogorov-arnold networks for time series: Bridging predictive power and interpretability, 2024.

[37] Yixuan Wang, Jonathan W. Siegel, Ziming Liu, and Thomas Y. Hou. On the expressiveness and spectral bias of kans, 2025.

[38] Hartmut Prautzsch, Wolfgang Boehm, and Marco Paluszny. *Bézier and B-Spline Techniques*. Springer, Berlin, Heidelberg, 2002.

[39] Zhijie Chen and Xinglin Zhang. Lss-skan: Efficient kolmogorov-arnold networks based on single-parameterized function, 2024.

[40] Sidharth SS, Keerthana AR, Gokul R, and Anas KP. Chebyshev polynomial-based kolmogorov-arnold networks: An efficient architecture for nonlinear function approximation, 2024.

[41] Victor Augusto Kich, Jair Augusto Bottega, Raul Steinmetz, Ricardo Bedin Grando, Ayano Yorozu, and Akihisa Ohya. Kolmogorov-arnold network for online reinforcement learning, 2024.

[42] Akash Kundu, Aritra Sarkar, and Abhishek Sadhu. Kanqas: Kolmogorov-arnold network for quantum architecture search. *EPJ Quantum Technology*, 11(1):76, 2024.

[43] Alexander M. Dalzell, Sam McArdle, Mario Berta, Przemyslaw Bienias, Chi-Fang Chen, András Gilyén, Connor T. Hann, Michael J. Kastoryano, Emil T. Khabiboulline, Aleksander Kubica, Grant Salton, Samson Wang, and Fernando G. S. L. Brandão. Quantum algorithms: A survey of applications and end-to-end complexities, 2023.

[44] Daan Camps, Lin Lin, Roel Van Beeumen, and Chao Yang. Explicit quantum circuits for block encodings of certain sparse matrices, 2023.

[45] András Gilyén, Yuan Su, Guang Hao Low, and Nathan Wiebe. Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, page 193–204, New York, NY, USA, 2019. Association for Computing Machinery.

[46] Guang Hao Low and Isaac L. Chuang. Hamiltonian simulation by qubitization. *Quantum*, 3:163, July 2019.

[47] R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca. Quantum algorithms revisited. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 454(1969):339–354, January 1998.

[48] Ikhsan Maulidi, Bonno Wibowo, Vina Apriliani, and Rofiqul Umam. The characteristics of the first kind of chebyshev polynomials and its relationship to the ordinary polynomials. *JTAM (Jurnal Teori dan Aplikasi Matematika)*, 5:323–331, 10 2021.

[49] Arthur G. Rattew and Patrick Rebentrost. Non-linear transformations of quantum amplitudes: Exponential improvement, generalization, and applications, 2023.

[50] Juan Carrasquilla and Roger G. Melko. Machine learning phases of matter. *Nature Physics*, 13(5):431–434, 2017.

[51] Xiao-Ming Zhang, Tongyang Li, and Xiao Yuan. Quantum state preparation with optimal circuit depth: Implementations and applications. *Physical Review Letters*, 129(23), November 2022.

[52] Costas Papachristou. The physical meaning of fermi-dirac statistics. 09 2024.

[53] Andrew M. Childs and Nathan Wiebe. Hamiltonian simulation using linear combinations of unitary operations. *Quantum Information and Computation*, 12(11 12), November 2012.

[54] Kerstin Beer. Quantum neural networks. 2022.

[55] T. Kato. Opensource software development kit. `https://github.com/Qaqarot`, 2018. Accessed: YYYY-MM-DD.

[56] Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. On the expressive power of deep neural networks, 2017.

[57] John P. Boyd. *Chebyshev and Fourier Spectral Methods*. Dover Publications, Mineola, NY, second revised edition edition, 2001.

[58] Lloyd N. Trefethen. *Approximation Theory and Approximation Practice*. SIAM, Philadelphia, PA, 2013.

[59] Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, Shahnawaz Ahmed, Vishnu Ajith, M. Sohaib Alam, Guillermo Alonso-Linaje, B. AkashNarayanan, Ali Asadi, Juan Miguel Arrazola, Utkarsh Azad, Sam Banning, Carsten Blank, Thomas R Bromley, Benjamin A. Cordier, Jack Ceroni, Alain Delgado, Olivia Di Matteo, Amintor Dusko, Tanya Garg, Diego Guala, Anthony Hayes, Ryan Hill, Aroosa Ijaz, Theodor Isacsson,

David Ittah, Soran Jahangiri, Prateek Jain, Edward Jiang, Ankit Khandelwal, Korbinian Kottmann, Robert A. Lang, Christina Lee, Thomas Loke, Angus Lowe, Keri McKiernan, Johannes Jakob Meyer, J. A. Montañez-Barrera, Romain Moyard, Zeyue Niu, Lee James O'Riordan, Steven Oud, Ashish Panigrahi, Chae-Yeun Park, Daniel Polatajko, Nicolás Quesada, Chase Roberts, Nahum Sá, Isidor Schoch, Borun Shi, Shuli Shu, Sukin Sim, Arshpreet Singh, Ingrid Strandberg, Jay Soni, Antal Száva, Slimane Thabet, Rodrigo A. Vargas-Hernández, Trevor Vincent, Nicola Vitucci, Maurice Weber, David Wierichs, Roeland Wiersema, Moritz Willmann, Vincent Wong, Shaoming Zhang, and Nathan Killoran. Pennylane: Automatic differentiation of hybrid quantum-classical computations, 2022.

[60] OpenAI. Chatgpt (june 2025 version). `https://chat.openai.com`, 2025. Large Language Model used as assistant for writing and code refinement.

[61] Kourosh Parand and Mehdi Delkhosh. The generalized fractional order of the chebyshev functions on nonlinear boundary value problems in the semi-infinite domain. *Nonlinear Engineering*, 6(3):229–240, 2017.

[62] Rui Wang, Yuesheng Xu, and Mingsong Yan. Hypothesis spaces for deep learning, 2024.

[63] Consorci de Serveis Universitaris de Catalunya (CSUC). Pirineus 3 Supercomputer. `https://www.csuc.cat/en/supercomputing`, 2025. Accessed: 2025-06-07.

[64] L. Sheridan, D. Maslov, and M. Mosca. Approximating fractional time quantum evolution. *Journal of Physics A: Mathematical and Theoretical*, 42(18):185302, 2009.

[65] Daan Camps and Roel Van Beeumen. Fable: Fast approximate quantum circuits for block-encodings. In *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*, page 104–113. IEEE, September 2022.

[66] Di Fang, Lin Lin, and Yu Tong. Time-marching based quantum solvers for time-dependent linear differential equations, 08 2022.

[67] John M. Martyn, Zane M. Rossi, Andrew K. Tan, and Isaac L. Chuang. Grand unification of quantum algorithms. *PRX Quantum*, 2:040203, Dec 2021.

Appendix A

SUPPLEMENTARY THEORY

A.1 Block-Encoding

Let A be a diagonal $J \times J$ matrix. A unitary A_{BE} is said to be a (α, a, ϵ)-Block Encoding [45] of A if

$$\|A - \alpha \left(\langle 0|_a \otimes I_n\right) A_{BE} \left(|0\rangle_a \otimes I_n\right)\| \leq \varepsilon \tag{A.1}$$

where $\alpha, \varepsilon \in \mathbb{R}_+$, $a \in \mathbb{N}$, are a scaling factor for A, an error tolerance and the number of auxiliary (anzilla) qubits used to form the Block-Encoding respectively. I_n is the n-qubit identity matrix where $h = \lceil \log_2 J \rceil$. U_{BE} will be a $(h + a)$-qubit unitary.

More generally, a Block Encoding A_{BE} is a unitary that embeds a user-specified matrix A as its top-left sub-block, as shown in Equation A.2

$$A_{BE} = \begin{bmatrix} A & * \\ * & * \end{bmatrix} \tag{A.2}$$

Hence, it is a form of quantum data encoding. The effect of applying A_{BE} to an $(a + n)$-quantum state $(|0\rangle_a \otimes |\phi\rangle_n)$ is the following: if once applied A_{BE} to the original state the first $|0\rangle_a$ are measured as $|1\rangle_a$ then the state of the other n qubits will be excatly $A |\phi\rangle_n$.

BEs can be used as a quantum data encoding technique, to encode any desired vector or matrix.

Basic BEs can serve as building blocks for constructing more complex BEs. That is, algebraic operations can be applied to BEs to derive new BEs [30]. This is the main principle used in QKAN to stack building blocks and construct layers.

There exist a huge number of ways to construct Block-Encoding unitaries [65] [44] [66] [45]. For this project purposes the function provided by PennyLane

qml.BlockEncode [[59]] has been used. It takes as input a matrix A, and constructs the unitary A_{BE} as:

$$A_{BE} = \begin{bmatrix} A & \sqrt{I - AA^\dagger} \\ \sqrt{I - A^\dagger A} & -A^\dagger \end{bmatrix} \tag{A.3}$$

Within this construction, A_{BE} is a $(1, h, 0)$-Block Encoding of A. This function is purely used for simulation purposes. It is not intended to be quantum hardware compatible since it requires to compute the square root of a product of matrices, which requires to compute and diagonalize such product, for which there is no efficient algorithm in a quantum computer with $\mathcal{O}(poly(h))$ unitaries [44].

A.2 Projector-Controlled Phase shift operators

Projector-Controlled Phase shift operators (PCPhase- shift)[67] (as defined in PennyLane [59]), are $n + a_x$-qubit unitaries that apply a complex phase $e^{i\phi}$ (i.e. an $Rz(\phi)$ gate) to the first n basis vectors of the input state and the phase $e^{-i\phi}$ to the remaining a_x basis vectors. Equation A.4 shows an example of this operator for $n = 1$ and $a_x = 1$.

$$\Pi_\phi = \begin{bmatrix} e^{i\phi} & 0 & 0 & 0 \\ 0 & e^{i\phi} & 0 & 0 \\ 0 & 0 & e^{-i\phi} & 0 \\ 0 & 0 & 0 & e^{-i\phi} \end{bmatrix} \tag{A.4}$$

Given a BE A_{BE} of a matrix A, if $\phi = 2n\pi$ with $n \in \mathbb{Z}$ then the operators simply select the matrix A:

$$A = \tilde{\Pi}_{2n\pi} A_{BE} \Pi_{2n\pi} \tag{A.5}$$

Where $\tilde{\Pi}_{2n\pi}$ acts on the row subspace of A and $\Pi_{2n\pi}$ on the column subspace.

A.3 Quantum Singular Value Transformation

Quantum Singular Value Transformation (QSVT) [45] is a framework used to design quantum algorithms. Its main objective is to transform a block encoding A_{BE} of a matrix A to the block encoding of $P(A)$ where P is an r-degree polynomial transformation of the singular values of A.

Supposing that A_{BE} is an (α, a, ε)-Block Encoding of A, then, QSVT will construct the $(\alpha, a + 1, 4r\sqrt{\varepsilon})$-Block Encoding:

$$P(A_{BE}) = \begin{bmatrix} P(A) & * \\ * & * \end{bmatrix} \tag{A.6}$$

The process to obtain $P(A_{BE})$ consists in interleaving the unitaries A_{BE}, $A_{BE}^{\dagger}$ and Projector-Controlled Phase-shift (PCP-shift) operators Π_{ϕ_i} and $\tilde{\Pi}_{\phi_j}$ (see A.2).

The calculation of $P(A_{BE})$ depends on the degree r of the polynomial implemented [67]:

For odd-degree polynomials:

$$P(A_{BE}) = \tilde{\Pi}_{\phi_1} A_{BE} \prod_{k=1}^{\frac{r-1}{2}} \Pi_{\phi_{2k}} A_{BE}^{\dagger} \tilde{\Pi}_{\phi_{2k+1}} A_{BE} \tag{A.7}$$

For even-degree polynomials:

$$P(A_{BE}) = \prod_{k=1}^{\frac{r}{2}} \Pi_{\phi_{2k-1}} A_{BE}^{\dagger} \tilde{\Pi}_{\phi_{2k}} A_{BE} \tag{A.8}$$

A.4 Linear Combination of Unitaries

Linear Combination of Unitaries (LCU) [53], is a technique that allows to form a unitary A defined as a linear combination of N gates U_k.

$$A = \sum_{k=0}^{N-1} \alpha_k U_k \tag{A.9}$$

Where $\alpha_k \in \mathbb{R}$. This operation is performed by two different quantum operators, Prepare (PREP) and Select (SEL). The first operator has per objective to prepare a quantum state with amplitudes depending on the coefficients α_k. The second operator selects the unitary to apply to each coefficient. Their effects on specific states are shown in Equation A.10.

$$\text{PREP}\,|0\rangle = \sum_k \sqrt{\frac{|\alpha_k|}{\lambda}}\,|k\rangle \qquad \text{SEL}\,|k\rangle\,|\psi\rangle = |k\rangle\,U_k\,|\psi\rangle \tag{A.10}$$

Where $\lambda = \sum_k |\alpha_k|$ is a normalization factor. The unitary that forms the entire circuit is then $\text{PREP}^{\dagger} \cdot \text{SEL} \cdot \text{PREP}$, and Equation A.11 shows the result of measuring the state $|0\rangle$ in the k anzilla qubits used for this operation.

$$\langle 0|\text{PREP}^{\dagger} \cdot \text{SEL} \cdot \text{PREP}|0\rangle\,|\psi\rangle = \frac{A}{\lambda}\,|\psi\rangle \tag{A.11}$$

When $|0\rangle$ is measured for the first k anzilla qubits, the state prepared is proportional to the desired matrix A up to normalization factor λ.

A.5 Hadamard Test

Considering the n-qubit state $|\psi\rangle$ and the n-qubit quantum gate U (that can be understood as an observable), a Hadamard Test creates a random variable whose expected value is $Re\,\langle\psi|\,U\,|\psi\rangle$.

As stated in 1.1.2, sometimes there is the need of obtaining a deterministic value from a quantum system. If the quantum gate U is such that we don't directly know its eigenvalues, the Hadamard Test allows to calculate the expected value of measuring the system. This is done by adding an anzilla register in equal superposition, controlling the application of the O gate when anzilla is in $|1\rangle$ state, undoing the equal superposition to return to the computational basis, and finally measuring the expected value of Z on the anzilla qubit. Circuit for doing this task is shown in Figure A.1.

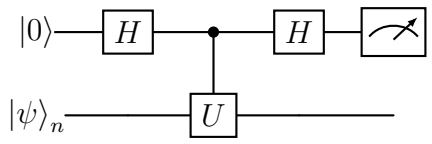


Figure A.1: Circuit implementing a Hadamard Test for measuring $Re\,\langle\psi|\,U\,|\psi\rangle$

A.6 Chebyshev Polynomials

Chebyshev polynomials [48], also called Chebyshev functions, form two sequences of orthogonal polynomials. These are denoted as $F_n(x)$ for the Chebyshev polynomials of the first kind and $U_n(x)$ for those of second kind. This project is entirely focused on Chebyshev polynomials of the first kind, hence, from now on, they will be referred to simply as *Chebyshev polynomials*.

A.6.1 Definition and properties

Definition

Chebyshev polynomials of degree r are defined as:

$$T_r(x) = \cos(r\theta) \tag{A.12}$$

where $x = \cos(\theta)$ and $x \in [-1, 1]$.

To derive the explicit form of these polynomials, it is useful to recall the trigonometric identity:

$$\begin{cases} \cos(a+b) = \cos(a)\cos(b) - \sin(a)\sin(b) \\ \cos(a-b) = \cos(a)\cos(b) + \sin(a)\sin(b) \end{cases} \tag{A.13}$$

Using this identity, the following recurrence relation is obtained:

$$\begin{aligned} \cos((r+1)\theta) + \cos((r-1)\theta) &= 2\cos(r\theta)\cos(\theta) \\ \Rightarrow \quad T_{r+1}(x) + T_{r-1}(x) &= 2xT_r(x) \\ \Rightarrow \quad T_{r+1}(x) &= 2xT_r(x) - T_{r-1}(x) \end{aligned} \tag{A.14}$$

This recurrence relation allows us to compute the Chebyshev polynomials iteratively. The first few polynomials are:

$$\begin{aligned} T_0(x) &= \cos(0) = 1 \\ T_1(x) &= \cos(\theta) = x \\ T_2(x) &= \cos(2\theta) = 2x^2 - 1 \\ T_3(x) &= \cos(3\theta) = 4x^3 - 3x \\ T_4(x) &= \cos(4\theta) = 8x^4 - 8x^2 + 1 \\ T_5(x) &= \cos(5\theta) = 16x^5 - 20x^3 + 5x \\ T_6(x) &= \cos(6\theta) = 32x^6 - 48x^4 + 18x^2 - 1 \end{aligned} \tag{A.15}$$

Orthogonality

These polynomials are orthogonal with respect to the weight function $w(x) = \frac{1}{\sqrt{1-x^2}}$ on the interval $[-1, 1]$:

$$\langle T_i(x), T_j(x) \rangle = \int_{-1}^{1} \frac{T_i(x)T_j(x)}{\sqrt{1-x^2}}\, dx = \frac{\pi}{c_i}\delta_{ij} \tag{A.16}$$

where $c_0 = 1$ and $c_n = 2$ for $n \geq 1$.

For that reason, they play a central role in approximation theory through the Chebyshev Series [58]:

$$f(x) = \sum_{k=0}^{\infty} a_k T_k(x) \tag{A.17}$$

defining a basis for spectral methods with the form: $\sum_{r=0}^{r=\infty} c_i x^r$.

A.6.2 Chebyshev approximation

Chebyshev approximation is a technique aimed to approximate a target function $f(x)$ defined on the interval $[-1, 1]$ as a truncated Chebyshev series:

$$f(x) \approx \sum_{r=0}^{d} a_r T_r(x) \tag{A.18}$$

Where $d > 0$ is the order of the approximation and coefficients a_k are represented as:

$$a_k = \frac{2}{\pi} \int_{-1}^{1} \frac{f(x)T_k(x)}{\sqrt{1-x^2}} dx \quad \text{for } k > 0, \qquad a_0 = \frac{1}{\pi} \int_{-1}^{1} \frac{f(x)}{\sqrt{1-x^2}} dx \tag{A.19}$$

When $f(x)$ is continuous and smooth, the Chebyshev series converges rapidly to $f(x)$, often exponentially. For functions that are not smooth, convergence is slower (e.g. $\sqrt{x}$) [58].

Within the CHEB-QKAN context, the coefficients a_k are the learnable weights.

A.7 Generalized Fractional-order Chebyshev Functions

The solution of some Differential Equations cannot be approximated through a polynomial basis, for example, the solution of the Ordinary Differential Equation:

$$4ff'' = 3x, \quad f(0) = f'(0) = 0, \quad f(t) = x^{\frac{3}{2}} \tag{A.20}$$

is $f(x) = x^{\frac{3}{2}}$. This may lack the ability to approximate non-polynomial functions through the Chebyshev approximation. For this reason, Kourosh Parand et al. defined a new basis for spectral methods as: $\sum_{r=0}^{r=\infty} c_i x^{r\alpha}$ [61].

Generalized Fractional-order of the Chebyshev Functions (GFCFs) [61] are a basis for spectral methods with the form:

$$\sum_{r=0}^{\infty} c_r x^{r\alpha} \tag{A.21}$$

where $\alpha \in \mathbb{R}_+$.

A.7.1 Definition and orthogonality property

GFCFs are defined over the interval $[0, \eta]$ as:

$$ {}_{\eta}FT_r^{\alpha}(x) = T_r\left(1 - 2\left(\frac{x}{\eta}\right)^{\alpha}\right) \tag{A.22} $$

where $\alpha, \eta \in \mathbb{R}_+$ and $x \in [0, \eta]$. This definition reveals the following analytical form:

$$ {}_{\eta}FT_r^{\alpha}(x) = \sum_{k=0}^{r} \beta_{r,k,\eta,\alpha}\, x^{\alpha k} \tag{A.23} $$

where $\beta_{r,k,\eta,\alpha} = (-1)^k \frac{r\, 2^{2k}(r+k-1)!}{(r-k)!(2k)!\, \eta^{\alpha k}}$ and $\eta_{0,k,\eta,\alpha} = 1$.

A.7.2 Ortogonality and GFCF approximation

GFCFs are orthogonal with respect to the weight function $w(x) = \frac{x^{\frac{\alpha}{2}-1}}{\sqrt{\eta^{\alpha}-x^{\alpha}}}$ in the interval $(0, \eta)$ [61]:

$$ \int_0^{\eta} {}_{\eta}FT_i^{\alpha}(x)\, {}_{\eta}FT_j^{\alpha}(x)\, w(x)\, dx = \frac{\pi}{2\alpha} c_i \delta_{ij} \tag{A.24} $$

where δ_{mn} is the Kronecker delta, $c_0 = 2$, and $c_n = 1$ for $n \geq 1$.

This property reveals that GFCFs are a suitable basis for the approximation of continuous functions on $[0, \eta]$. Analogously to the Chebyshev expansion, GFCFs can construct the following series:

$$ f(x) = \sum_{r=0}^{\infty} {}_{\eta}FT_r^{\alpha}(x) a_r \tag{A.25} $$

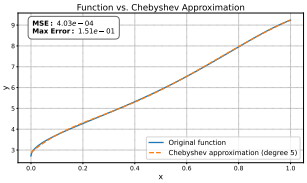
where the coefficients a_r are obtained as:

$$ a_r = \frac{2\alpha}{\pi c_r} \int_0^{\eta} {}_{\eta}FT_r^{\alpha}(x)\, y(x)\, w(x)\, dx \tag{A.26} $$

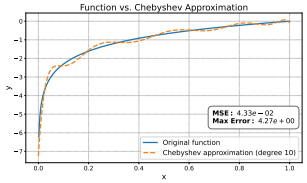
By truncating the previous series up to a maximum degree d one can obtain an approximation of $f(x)$ of order d as:

$$ f(x) \approx \sum_{r=0}^{d} {}_{\eta}FT_r^{\alpha}(x) a_r \tag{A.27} $$

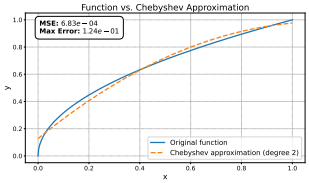
The GFCF approximation is able to outperform the Chebyshev approximation:



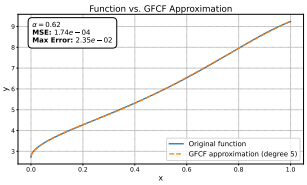
(a) Chebyshev approximation of function 1



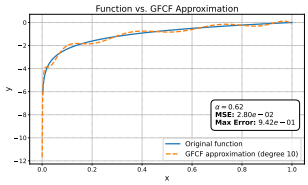
(b) Chebyshev approximation of function 2



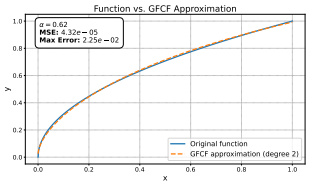
(c) Chebyshev approximation of function 3



(d) GFCF approximation of function 1



(e) GFCF approximation of function 2



(f) GFCF approximation of function 3

Figure A.2: (Row 1) Chebyshev approximations, (Row 2) GFCF approximations. (a & d) Function 1: $f(x) = \exp\left(\sin\left(\sqrt{x}\right) + \cos(x) + \sin(x^2)\right)$, (b & e) Function 2: $f(x) = log(x)$, (c & f) Function 3: $f(x) = \sqrt{x}$.

A.8 PennyLane Framework

A.8.1 What is PennyLane?

PennyLane [59] is a quantum computing framework developed by Xanadu that enables the construction, simulation, and training of hybrid quantum-classical models. One of its key features is the seamless integration of quantum circuits with classical machine learning libraries such as *NumPy*, *TensorFlow*, and *PyTorch*.

PennyLane handles the execution of quantum operations, such as running circuits and computing their gradients, and forwards this information to the classical framework. This enables the creation of smooth and efficient quantum-classical pipelines, where quantum computations can be embedded directly within classical optimization loops.

A.8.2 Devices

The framework supports a wide range of quantum devices, including local simulators and hardware backends from different providers, allowing developers to prototype on simulators and then transition to real quantum hardware with minimal changes to the code. This flexibility makes PennyLane a powerful tool for

building and testing quantum-enhanced machine learning models in both simulated and real environments.

A.8.3 PennyLane qnodes

PennyLane introduces the concept of a *quantum node* (qnode), which wraps a quantum circuit as a differentiable Python function. A QNode combines a quantum device with a circuit-defining function. When executed, it behaves like any regular Python function and can be seamlessly integrated into classical training pipelines.

A qnode acts as a variational quantum circuit that can be optimized through different classical optimizers that are integrated within the PennyLane framework by leveraging other classical ML libraries.

A.8.4 Circuit wires

In PennyLane, quantum circuits are defined through *wires*, which represent the individual qubits on which quantum gates are applied. Each wire is identified by an index or a user-defined label, allowing for flexible and readable circuit definitions. When constructing a circuit, quantum operations (such as unitary gates, rotations, or measurements) are applied explicitly to specific wires, determining how the quantum state evolves throughout the computation. Figure A.3 shows a Quantum Circuit with 3 wires.

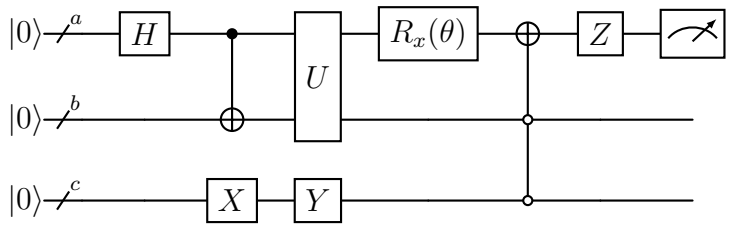


Figure A.3: Quantum circuit containing 3 wires, named in order: a, b, and c.

Wires are used to control the logical layout of the quantum circuit and must be consistent with the number of qubits available on the selected device. The number and labeling of wires are specified when initializing the device, and any mismatch between the circuit definition and the declared wires typically results in a runtime error.

Appendix B

SUPPLEMENTARY FIGURES

B.1 Quantum gates

Graphical representation of different Quantum Gates [4].

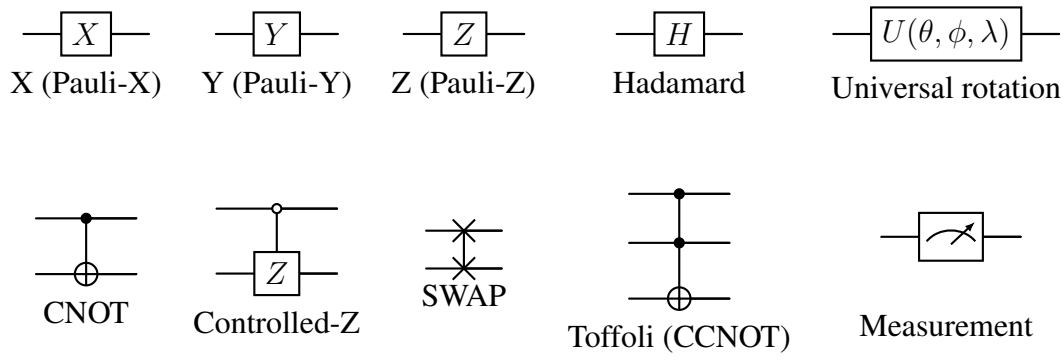


Figure B.1: Graphical representation of fundamental quantum gates [4] acting on 1 and 2 qubits, including Pauli operators, the Hadamard gate, the universal rotation gate $U(\theta, \phi, \lambda)$, a measurement gate, and controlled multi-qubit gates. Control qubits with a closed black dot are conditioned on $|1\rangle$, while control qubits with a white dot are conditioned on $|0\rangle$ (as used for the controlled-Z gate).

Appendix C

SUPPLEMENTARY EQUATIONS

C.1 Introduction

Single-qubit rotation gates definition

$$R_z(\theta) = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix}, \quad R_x(\alpha) = \begin{bmatrix} \cos\left(\frac{\alpha}{2}\right) & -i\sin\left(\frac{\alpha}{2}\right) \\ -i\sin\left(\frac{\alpha}{2}\right) & \cos\left(\frac{\alpha}{2}\right) \end{bmatrix}$$
$$R_y(\beta) = \begin{bmatrix} \cos\left(\frac{\beta}{2}\right) & -\sin\left(\frac{\beta}{2}\right) \\ \sin\left(\frac{\beta}{2}\right) & \cos\left(\frac{\beta}{2}\right) \end{bmatrix} \tag{C.1}$$

Multi-qubit rotation gates definition:

$CNOT$ gate definition:

$$CNOT = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{C.2}$$

Appendix D

SUPPLEMENTARY TABLES

D.1 Introduction

$CNOT$ gate action over a 2-qubit state:

Input State	Output State after CNOT
$\|00\rangle$	$\|00\rangle$
$\|01\rangle$	$\|01\rangle$
$\|10\rangle$	$\|11\rangle$
$\|11\rangle$	$\|10\rangle$

Table D.1: Action of the CNOT gate on computational basis states.

D.2 Results

D.2.1 Experimental Results

Training time distribution across runs for Classification task:

Model	Mean	Median	Std	Min	Max
Plain-CHEB	843.63	850.74	92.69	693.76	1008.10
Plain-Flex	1136.16	1143.17	82.29	1016.24	1230.41
GFCF-CHEB-0	787.74	725.47	122.30	681.04	997.26
GFCF-CHEB-1	884.34	872.67	70.30	778.78	975.47
GFCF-Flex-0	1101.39	1062.50	145.79	949.73	1391.35
GFCF-Flex-1	1159.69	1180.27	135.00	955.32	1334.98

Table D.2: Distribution training times (in seconds) per run for tested models within Classification task. (Numerical values)

Training time distribution across runs for Exponential Regression task:

Model	Mean	Median	Std	Min	Max
GFCF-CHEB-0	787.74	725.47	122.30	681.04	997.26
GFCF-CHEB-1	884.34	872.67	70.30	778.78	975.47
GFCF-Flex-0	1101.39	1062.50	145.79	949.73	1391.35
GFCF-Flex-1	1159.69	1180.27	135.00	955.32	1334.98
Plain-CHEB	843.63	850.74	92.69	693.76	1008.10
Plain-Flex	1136.16	1143.17	82.29	1016.24	1230.41

Table D.3: Distribution of training times (in seconds) per run for the tested models within the Exponential Regression task.

Training time distribution across runs for Polynomial Regression task:

Model	Mean	Median	Std	Min	Max
GFCF-CHEB-0	1933.52	1910.77	83.61	1880.24	2154.33
GFCF-CHEB-1	2375.62	2338.51	171.07	2200.31	2652.69
GFCF-Flex-0	2324.97	2315.89	256.20	2071.77	2660.38
GFCF-Flex-1	3037.14	3031.23	90.53	2830.77	3153.80
Plain-CHEB	1930.74	1850.38	160.91	1786.20	2175.38
Plain-Flex	2072.89	2066.55	20.71	2048.75	2105.85

Table D.4: Distribution of training times (in seconds) per run for the tested models within the Polynomial Regression task.