Fully Quanvolutional Networks for Time Series Classification

Nabil Anan Orka
n.orka@uq.edu.au
The University of Queensland
Brisbane, Australia

Ehtashamul Haque
ehtashamul.haque@bracu.ac.bd
BRAC University
Dhaka, Bangladesh

Md. Abdul Awal
m.awal@uq.edu.au
The University of Queensland
Brisbane, Australia

Mohammad Ali Moni*†
mmoni@csu.edu.au
Charles Sturt University
Orange, Australia

Abstract

Despite the advancements in quantum convolution or quanvolution, challenges persist in making quanvolution scalable, efficient, and applicable to multi-dimensional data. Existing quanvolutional networks heavily rely on classical layers, with minimal quantum involvement due to inherent limitations in current quanvolution algorithms. Moreover, the application of quanvolution in the domain of 1D data remains largely unexplored. To address these limitations, we propose a new quanvolution algorithm–Quanv1D–capable of processing arbitrary-channel 1D data, handling variable kernel sizes, and generating a customizable number of feature maps, along with a classification network–fully quanvolutional network (FQN)–built solely using Quanv1D layers. Quanv1D is inspired by the classical Conv1D and stands out from the quanvolution literature by being fully trainable, modular, and freely scalable with a self-regularizing feature. To evaluate FQN, we tested it on 20 UEA and UCR time series datasets, both univariate and multivariate, and benchmarked its performance against state-of-the-art convolutional models (both quantum and classical). We found FQN to outperform all compared models in terms of average accuracy while using significantly fewer parameters. Additionally, to assess the viability of FQN on real hardware, we conducted a shot-based analysis across all the datasets to simulate statistical quantum noise and found our model robust and equally efficient.

CCS Concepts

• **Computing methodologies** → **Machine learning algorithms**; **Supervised learning by classification**.

Keywords

quanvolution, quantum deep learning, time series

ACM Reference Format:
Nabil Anan Orka, Ehtashamul Haque, Md. Abdul Awal, and Mohammad Ali Moni. 2025. Fully Quanvolutional Networks for Time Series Classification. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '25), August 3–7, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3711896.3736972

KDD Availability Link:
The source code of this paper has been made publicly available at https://doi.org/10.5281/zenodo.15486109.

*Corresponding author
†Also with Washington University of Science and Technology.

KDD '25, Toronto, ON, Canada.

ACM ISBN 979-8-4007-1454-2/2025/08
https://doi.org/10.1145/3711896.3736972

1 Introduction

As Moore's law approaches its physical limits in a classical setting, a global shift is underway toward quantum computing to overcome the limitations of classical computers [32]. With companies like IBM and Google actively testing and developing quantum computers [1, 6], researchers are exploring the migration of classical algorithms to the quantum domain in anticipation of a future where quantum computing is commercially accessible. Given the fundamentally different operating paradigms of quantum and classical systems, not all classical algorithms can be directly implemented on quantum computers [32]. However, achieving a semblance of performance and functionality is highly desirable in most cases.

One prominent family of algorithms for which quantum equivalents are being actively developed is machine learning (ML). ML algorithms are computationally expensive, making them ideal candidates to benefit from the "quantum advantage" that quantum computing promises. As a result, quantum machine learning (QML) has emerged as a promising field, with progress in quantum adaptations of support vector machines [27, 37] and neural networks [45, 46]. Quantum convolution, sometimes referred to as "quanvolution" in some cases (the demarcation is elaborated in the next section), has also undergone some development in recent years [8, 17, 19, 30]. However, compared to their classical counterparts, existing quanvolution modules often fall short in terms of performance and utility. As such, further research and the development of more robust quanvolutional frameworks are necessary.

This paper introduces a novel 1D quanvolutional framework that demonstrates performance on par with classical convolution (and, in some cases, outperforms it) while offering greater scalability and modularity than existing quanvolutional models.

1.1 Background

In the literature, there are two variants of quantum convolution: circuit-based and kernel-based. The circuit-based variant mimics Conv2D and pooling layers with different quantum operations to turn a convolutional neural network (CNN) into a quantum circuit [8, 19]. However, unlike a typical convolution operation, this variant

works with the entire flattened input instead of individual input patches. Although the entire process can be executed on a quantum computer, the scalability of circuit-based quantum convolution compared to classical CNNs remains open for debate. For instance, this variant struggles to encode large, complex image datasets in this noisy intermediate-scale quantum (NISQ) era. Additionally, the reliance of neural networks on nonlinearities conflicts with quantum mechanics incapable of nonlinear operations.

In contrast, kernel-based quantum convolution, also known as quanvolution, functions similarly to a single classical convolutional layer [17]. Essentially, the quanvolution operation results from quantum circuits substituting for typical filters or kernels inside a convolutional layer. This approach offers greater flexibility than the circuit-based variant, enabling the development of hybrid models and training schemes. In a classical-quantum hybrid model, a quantum computer deals with the primary computations–interactions between input patches and filters–while classical computers handle tasks like processing loss values and applying nonlinear activations [35]. Owing to its versatility, the algorithm's introduction remains a staple in quantum-based computer vision with a wide range of applications [22, 34, 49, 51, 53].

Our study focuses on the kernel-based quantum convolution, or quanvolution, and aims to address various limitations outlined in the following section.

1.2 Motivation

Quanvolutional neural networks are implemented as hybrid systems, where learning is achieved through the combined effort of quantum and classical layers. In such architectures, the initial convolution layer in models like LeNet-5 [25] is replaced with a quanvolutional layer, while the rest of the network remains unchanged. Even when such models perform well, questions arise about the valid contribution of quanvolutional layers—whether they play a meaningful role or if the classical layers are doing the heavy lifting. Additionally, with only one quantum layer, as is typical in quanvolutional neural networks, it is uncertain if the model can truly achieve the "quantum advantage."

The original quanvolution algorithm was initially designed to work only with single-channel image patches [17]. Although some studies have managed to extend its application to RGB or three-channel images [21, 40], the algorithm still lacks the ability to process data with an arbitrary number of channels. This limitation restricts the placement of a quanvolutional layer deeper within the network, as the number of channels typically increases in later layers (observed in architectures like ResNet [16], EfficientNet [47], Inception [44], and DenseNet [18]). Moreover, existing quanvolutional algorithms can only produce a limited and fixed number of feature maps because they rely on just one filter. In contrast, in classical convolution, a layer utilizes multiple kernels or filters to extract varied but pertinent patterns from the same input, improving the model's generalization ability [24]. Additionally, owing to the reliance on angle embedding or similar linear encoding, the existing quanvolutional algorithms have to keep their kernel size small, or the algorithm gets too computationally expensive. A smaller kernel restricts the effective receptive fields [29], hindering the layer from capturing broader contextual features. In the literature, the limited number of output features resulting from a single filter has been partially addressed by mapping the output of each wire to separate output channels. However, the number of output features remains relatively low due to the small kernel sizes, as the kernel size influences the number of wires in the circuit, consequently affecting the number of feature maps. Furthermore, most implementations do not use trainable weights for the quanvolutional layer, limiting proper representation learning.

The 1D variant of quanvolution has received even less attention in existing research. We found only two articles on 1D quanvolution [39, 42]. Since these two studies drew inspiration from the original 2D quanvolution, the limitations of the 2D variant, mainly related to scalability, limited feature maps, and rigid kernel size, carry over to the 1D variant. For instance, in the implementation of Rivera-Ruiz et al. [39], the authors replaced the 2D kernel of Henderson et al. [17] with a fixed-length, trainable 1D kernel and kept the rest of the model as is by changing 2D convolutional layers to 1D convolutional layers. Soltani et al. [42] used the same 1D quanvolutional algorithm as Rivera-Ruiz et al. [39] while keeping the weights of the quantum layer untrainable and replacing the rest of the model with an echo state network. Aside from these two, some studies also worked with quanvolution applied to 1D data [26, 36, 40, 43, 51]. However, instead of adapting to a 1D variant of quanvolution, they converted the 1D data into visual representations such as scalograms or spectrograms and then applied 2D quanvolution.

Given all the restrictions and shortcomings of existing quanvolutional algorithms, our motivation for this work was to create a learnable, scalable, and modular quanvolutional layer that works like modern convolutional layers and gives users the freedom to choose the kernel size, the number of output feature maps, and other common hyperparameters. We aimed to ensure that this layer could be seamlessly integrated into any neural network, provided the input dimensions were compatible, mirroring the versatility of conventional convolution. In this study, we opted to work with 1D quanvolution to establish a strong foundational understanding before advancing to the 2D variant. Moreover, given the constraints of simulating quantum computing on classical hardware, 1D data presents a practical advantage as it involves less computational overhead compared to 2D. We evaluated our 1D quanvolutional algorithm on 1D time series data, including univariate and multivariate datasets, to analyze the differences between quanvolution and convolution in representation learning.

1.3 Contributions

As a main contribution, this study introduces the Quanv1D layer, a quantum analog to the Conv1D layer. Similar to Conv1D, Quanv1D supports multichannel data, variable kernel lengths, and adaptable feature map generations. While designing this layer, we prioritized efficiency and practicality, considering the limitations of current quantum hardware and aiming to minimize computational costs. For instance, based on the input requirements and desired output, the layer adjusts itself by either using a higher number of qubits with fewer circuits or reducing the number of qubits while increasing the number of filters. Additionally, we reduced the qubit usage with amplitude embedding for scalable data encoding, which

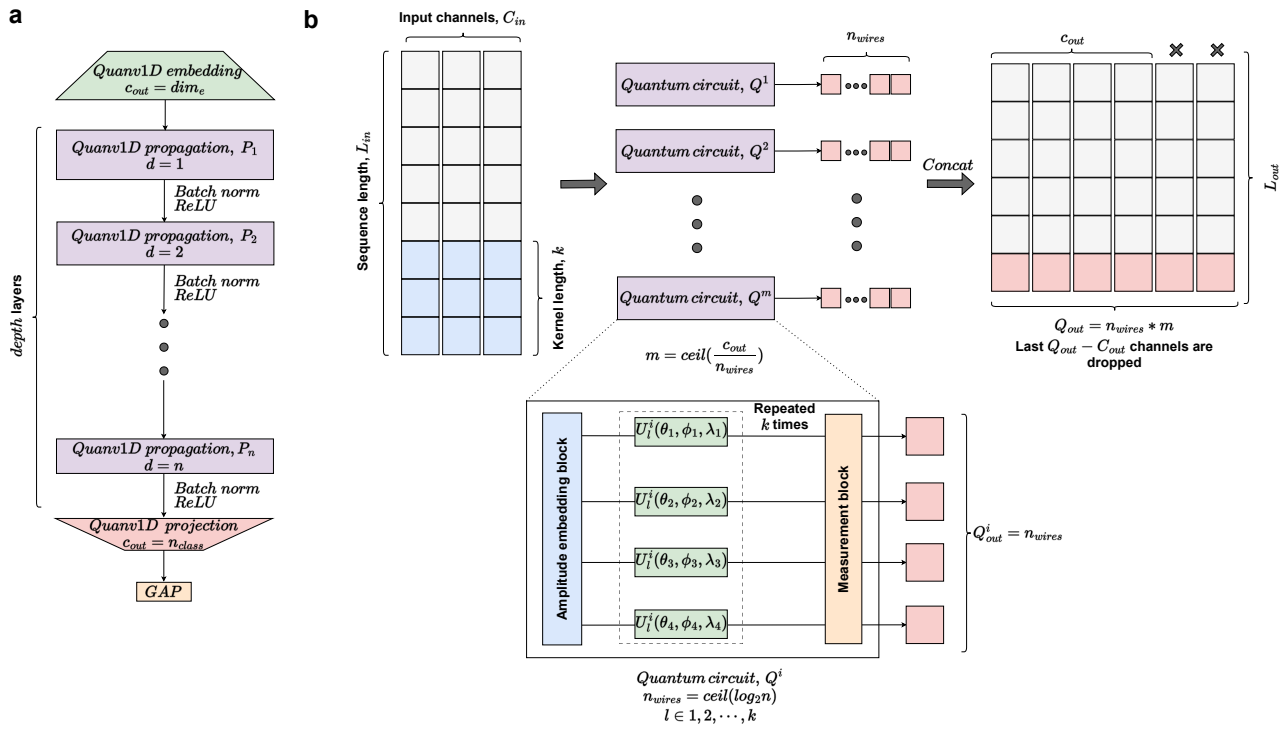

Figure 1: (a) FQN's overall architecture. It has three main parts–embedding, propagation, and projection–consisting of Quanv1D layers. We included batch normalization and ReLU activation in the architecture for learning stability and non-linearity. GAP denotes global average pooling. (b) Workflow inside the Quanv1D layer. Unlike a conventional filter found in a convolutional layer, Quanv1D uses quantum circuits to generate feature maps.

resulted in a single filter requiring $log_2(C_{in} * k)$ qubits instead of $C_{in} * k$ qubits stemming from linear mapping methods. Here, C_{in} represents the input channel size, and k represents the kernel length. Furthermore, thanks to our carefully chosen ansatz, Quanv1D exhibits a self-regularizing property that enhances training performance.

Our secondary contribution lies in designing an efficient quanvolutional neural network for time series classification. We deviated from the standard design where there is an overreliance on classical layers and instead built a fully quanvolutional network (FQN) built only with our Quanv1D layer. This design demonstrates the potential of stacked quantum layers to enhance representation learning in temporal data without relying on classical layers. Moreover, FQN is very lightweight, requiring substantially fewer trainable parameters than its quantum and classical counterparts.

The following section provides detailed technical explanations of Quanv1D and FQN's design specifics. In Appendix A, we discuss some quantum computing basics to help readers follow along.

2 Method

2.1 Quanv1D

In general, Conv1D accepts input in the form (N, C_{in}, L_{in}), where N represents the batch size, C_{in} is the number of input channels or dimensions, and L_{in} refers to the sequence length. The output shape is (N, C_{out}, L_{out}), where C_{out} is a user-defined parameter specifying the number of output channels following the quanvolution operation. The value of L_{out} is computed using the following equation:

$$L_{out} = \left\lfloor \frac{L_{in} + 2 \times p - d \times (k-1) - 1}{s} + 1 \right\rfloor \tag{1}$$

Here, k stands for the kernel size, s for the stride, p for the padding on both sides of the input, and d for the spacing between kernel points. Quanv1D has been designed to mimic a Conv1D layer and follows the same patching operations [7]. For this reason, we use the hyperparameters presented in Equation (1) to determine the patches that will be input to the quanvolutional filters. These hyperparameters, such as the kernel length, also affect quantum-related calculations inside the layer, like how many quantum filters there are, how many qubits are in a single filter, how many unitary operations are in an ansatz, and so on.

This study uses amplitude embedding to convert the classical information from the extracted patches into a quantum feature space [41]. This method encodes 2^n features into the amplitude vector of n qubits, as shown in the following equation:

Table 1: Descriptions of the datasets used in the study.

Code	Dataset	Sample	Length	Dim	Type	Domain
D1	Chinatown	363	24	1	Traffic	Urban planning
D2	SharePriceIncrease	1931	60	1	Financial	Stock market
D3	SyntheticControl	600	60	1	Simulated	Synthetic data analysis
D4	PhalangesOutlinesCorrect	2658	80	1	Image	Osteology
D5	ECG200	200	96	1	ECG	Cardiovascular diagnostics
D6	PowerCons	360	144	1	Device	Smart grid
D7	ToeSegmentation2	166	343	1	Motion	Biomechanics
D8	DiatomSizeReduction	322	345	1	Image	Microbiology
D9	Earthquakes	461	512	1	Sensor	Seismology
D10	InsectEPGRegularTrain	311	601	1	EPG	Entomology
D11	StarLightCurves	9236	1024	1	Sensor	Astronomy
D12	NerveDamage	204	1500	1	EMG	Neurophysiology
D13	BinaryHeartbeat	409	18530	1	Audio	Cardiovascular diagnostics
D14	Epilepsy	275	206	3	Sensor	Human activity recognition
D15	EthanolConcentration	524	1751	3	Spectroscopy	Chemical analysis
D16	Blink	950	510	4	EEG	Brain-computer interface
D17	SelfRegulationSCP2	380	1152	7	EEG	Brain-computer interface
D18	HandMovementDirection	234	400	10	EEG	Brain-computer interface
D19	FingerMovements	416	50	28	EEG	Brain-computer interface
D20	MotorImagery	378	3000	64	EEG	Brain-computer interface

ECG: electrocardiogram; EPG: electrical penetration graph; EMG: electromyogram; EEG: electroencephalogram

$$|\psi\rangle = \sum_{i=1}^{2^n} \alpha_i |i\rangle \tag{2}$$

In this equation, α_i are the elements of the amplitude vector α, and $|i\rangle$ represent the computational basis states. Each quanvolutional filter takes an input of $C_{in} \times k$ features, requiring $n = \lceil \log_2(C_{in} \times k) \rceil$ qubits for encoding. Before encoding, the features are normalized using $\sqrt{softmax(\alpha)}$ to ensure $|\alpha|^2 = 1$. Additionally, if $C_{in} \times k$ is smaller than 2^n, the features are padded with zeros after normalization to match the required dimension size. We chose amplitude embedding over the usual linear mapping method, such as angle encoding, to reduce qubit usage and achieve compact data representation. This approach ensures the modularity we tried to achieve, which, otherwise, would be difficult to achieve with angle encoding due to its requirement for an impractically large number of qubits (even in simulations).

The matrix form of the chosen unitary operator, U, is given by:

$$U(\theta, \phi, \lambda) = \begin{pmatrix} \cos(\theta\frac{\pi}{2}) & -e^{i\lambda}\sin(\theta\frac{\pi}{2}) \\ e^{i\phi}\sin(\theta\frac{\pi}{2}) & e^{i(\phi+\lambda)}\cos(\theta\frac{\pi}{2}) \end{pmatrix} \tag{3}$$

Here, θ and λ are trainable, while ϕ is fixed but initialized randomly. This is because, although ϕ is essential for introducing phase shifts within the circuit, its gradient during parameter updates is theoretically derived to be zero (refer to Appendix F). Consequently, it remains static throughout the optimization process.

For an n-qubit circuit, the unitary operator is applied to each qubit, forming a layer of unitaries, and this layer is repeated k times. The total unitary operations can be expressed as follows:

$$U_{\text{total}} = \prod_{l=1}^{k} \left(\bigotimes_{i=1}^{n} U_{il}(\theta_{il}, \phi_{il}, \lambda_{il}) \right) \tag{4}$$

Here, $U_{il}(\theta_{il}, \phi_{il}, \lambda_{il})$ represents the unitary operation acting on the i-th qubit in the l-th layer. Let $|\psi_o\rangle$ represent the quantum state of the circuit after the unitary operations have been applied. The following equations describe our decoding process:

$$E_i = \langle \psi_o | Z_i | \psi_o \rangle \tag{5}$$

$$Z_i = I^{\otimes(i-1)} \otimes Z \otimes I^{\otimes(n-i)} \tag{6}$$

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{7}$$

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{8}$$

We measure each qubit in the circuit using Equation (5), where E_i represents the expectation value and Z_i is the observable for the i-th qubit. Although the circuit or filter's operations and calculations occur in the complex domain, the resulting expectation values are real and fall within the range of $[-1, 1]$. For instance, an expectation value close to -1 suggests a high probability of the qubit being in the $|1\rangle$ state and vice versa.

Each expectation value is mapped to a distinct output channel. A filter with n qubits will produce n feature maps, and the total number of filters required is $\left\lfloor \frac{C_{\text{out}}+n-1}{n} \right\rfloor$. However, if the total number of feature maps exceeds the user-defined number of output channels, we discard the extra maps, retaining only C_{out} feature maps. Finally, we add a bias term to the generated features for each output channel, just like in a classical convolutional layer.

2.2 FQN

The FQN architecture is designed to be parameter-efficient, built exclusively with Quanv1D layers, as shown in Fig. 1. It consists of three primary components: embedding, propagation, and projection. In summary, the embedding layer transforms the input into a

Table 2: Classification performance across datasets. Each experiment is repeated five times. We present the mean and standard deviation for test accuracy (%). Bold indicates the best performance.

Dataset	FQN	FCN	QuanvNet*	ESQN*	OS-CNN	ROCKET	InceptionTime	ModernTCN	TimesNet†
D1	96.4 (1.2)	96.4 (0.8)	93.2 (5.1)	63.8 (17.1)	97.0 (0.6)	97.3 (0.0)	96.4 (1.2)	97.5 (0.6)	**98.1 (0.8)**
D2	59.5 (1.0)	59.0 (1.1)	58.0 (2.6)	48.2 (5.5)	61.9 (0.9)	57.6 (0.5)	**62.3 (1.8)**	61.3 (0.9)	62.0 (1.1)
D3	99.0 (0.4)	99.3 (0.4)	96.0 (2.0)	39.5 (4.8)	**100.0 (0.0)**	99.8 (0.4)	99.5 (0.5)	96.8 (0.7)	99.3 (0.4)
D4	78.7 (1.0)	82.0 (2.0)	70.9 (1.8)	61.2 (4.2)	81.0 (0.8)	**82.6 (1.1)**	80.5 (0.8)	82.5 (1.4)	81.3 (0.6)
D5	80.5 (4.1)	74.5 (2.7)	76.0 (7.2)	63.5 (4.5)	82.5 (3.1)	**84.0 (1.4)**	80.5 (2.1)	82.0 (3.3)	79.0 (3.8)
D6	93.3 (1.2)	99.2 (0.8)	87.2 (3.3)	62.8 (28.4)	92.2 (2.1)	95.8 (1.0)	97.2 (1.0)	**99.4 (0.8)**	99.2 (1.2)
D7	**95.8 (1.7)**	90.3 (3.3)	81.8 (8.6)	66.1 (6.9)	93.3 (3.3)	**95.8 (1.7)**	90.3 (2.5)	76.4 (5.0)	77.6 (8.2)
D8	98.1 (1.3)	**100.0 (0.0)**	89.1 (5.3)	59.4 (20.3)	92.2 (7.2)	**100.0 (0.0)**	98.8 (2.0)	**100.0 (0.0)**	99.4 (0.9)
D9	75.7 (2.3)	73.3 (4.5)	66.5 (12.3)	71.7 (9.3)	**76.1 (2.4)**	75.2 (2.1)	67.6 (4.7)	70.2 (3.7)	75.2 (2.7)
D10	**100.0 (0.0)**	**100.0 (0.0)**	**100.0 (0.0)**	49.4 (13.8)	**100.0 (0.0)**	**100.0 (0.0)**	**100.0 (0.0)**	**100.0 (0.0)**	**100.0 (0.0)**
D11	96.9 (0.3)	**97.6 (0.2)**	95.6 (0.6)	42.0 (6.2)	97.5 (0.3)	97.4 (0.2)	97.0 (0.3)	92.1 (1.9)	92.7 (0.5)
D12	99.5 (1.1)	99.5 (1.1)	98.5 (1.1)	28.8 (4.7)	**100.0 (0.0)**	97.6 (0.0)	**100.0 (0.0)**	55.1 (5.6)	78.5 (7.0)
D13	72.7 (1.1)	**74.1 (3.4)**	73.7 (4.1)	63.9 (8.8)	67.8 (1.4)	58.3 (2.0)	72.7 (3.5)	60.0 (6.8)	–
D14	99.3 (1.0)	98.9 (1.0)	–	–	99.6 (0.8)	**100.0 (0.0)**	**100.0 (0.0)**	94.5 (3.6)	86.2 (4.2)
D15	30.9 (2.1)	26.1 (3.1)	–	–	36.0 (1.7)	**36.2 (0.7)**	33.0 (4.5)	26.7 (3.8)	28.6 (2.7)
D16	99.2 (0.3)	99.6 (0.2)	–	–	**99.9 (0.2)**	99.8 (0.3)	99.5 (0.4)	99.5 (0.0)	98.3 (0.8)
D17	**56.8 (2.4)**	44.7 (0.9)	–	–	53.2 (2.2)	56.1 (1.2)	53.4 (5.8)	51.1 (5.4)	49.2 (4.2)
D18	35.7 (3.5)	31.1 (6.3)	–	–	40.0 (6.6)	39.6 (2.4)	35.3 (4.1)	48.1 (7.3)	**54.9 (2.8)**
D19	56.1 (2.8)	55.7 (2.9)	–	–	53.0 (3.5)	56.4 (1.0)	52.5 (2.3)	**57.4 (1.1)**	53.7 (4.2)
D20	**52.4 (1.1)**	51.1 (6.6)	–	–	51.6 (3.1)	43.4 (0.0)	49.2 (2.4)	48.4 (7.1)	46.3 (5.2)
Average	**78.8**	77.6	–	–	78.7	78.6	78.3	74.9	–

*The results for QuanvNet and ESQN are incomplete because these models can handle univariate data only.

†Due to its embedding scheme, TimesNet cannot handle sequence lengths larger than 5000. As such, TimesNet's D13 performance is incomplete.

high-dimensional feature space, the propagation layers employ dilation [52] to iteratively improve feature representations with longer receptive fields, and the projection layer transfers these enhanced features to a desired output space.

The embedding layer expands the raw input data into a learned embedding dimension, dim_e, which helps aggregate local information and prepares the data for more complex hierarchical feature extraction. For long sequences, the kernel size k_e can remain large without shortening the sequence length, as padding is automatically adjusted using $\left\lfloor \frac{k_e}{2} \right\rfloor$. However, this adjustment is only applicable to kernels with odd sizes. In addition, strides can reduce the input sequence length if necessary. For example, setting $s_e = 3$ will shorten the embedded feature sequence to one-third of the original input length.

After the embedding stage, the propagation layers form the core of the architecture. There are $depth$ propagation layers, each composed of a Quanv1D layer, followed by batch normalization and ReLU activation. These layers improve the embedded representation over time by reducing the internal covariate shift and adding nonlinearity to the data. The dilation rate, $d_i = i$ (where i is the layer index), linearly increases the network's receptive field as the depth grows. This allows the model to capture dependencies at multiple scales–from local interactions in the lower layers to long-range dependencies in the deeper layers–without increasing the number of parameters. In the propagation layers, no padding was applied, and s_{prop} was set to one. However, the kernel size, k_{prop}, was adjusted based on the specific dataset.

Following the propagation layers, the projection layer maps the multi-scale, transformed features to a space corresponding to the target classes. This allows the network to produce class-specific feature maps. For this layer, the kernel size is set to $k_{proj} = 1$, with a stride of $s_{proj} = 1$, no padding ($p_{proj} = 0$), and a dilation rate of $d_{proj} = 1$. We then apply global average pooling (GAP) across the channel dimension and pass the pooled features to a softmax classifier for final prediction. Using GAP instead of linear layers helps prevent overfitting by reducing the number of trainable parameters and acts as a form of regularization.

3 Experiments

3.1 Datasets and Models

To evaluate the performance of FQN (and Quanv1D), we utilized 20 datasets from the UEA and UCR time series archives [3, 10]. We randomly selected the datasets but ensured a broad spectrum of practical applications. As such, the selected binary and multi-class datasets cover 15 different fields, with inputs that have different sizes (up to 64 channels) and lengths (up to 18530). Table 1 provides a detailed description of these datasets, and Appendix C outlines the selection process used in this study.

Our goal was to compare FQN to a fully convolutional network (FCN) with the same architecture and hyperparameters but with classical convolutional layers. This comparison helped us examine the differences between Quanv1D and Conv1D regarding training and inference. We also added two 1D quanvolutional networks, QuanvNet [39] and ESQN [42], in the mix. To assess FQN's benchmarking potential, we included ModernTCN [28], the current state-of-the-art (SOTA) in time series classification, in the comparison,

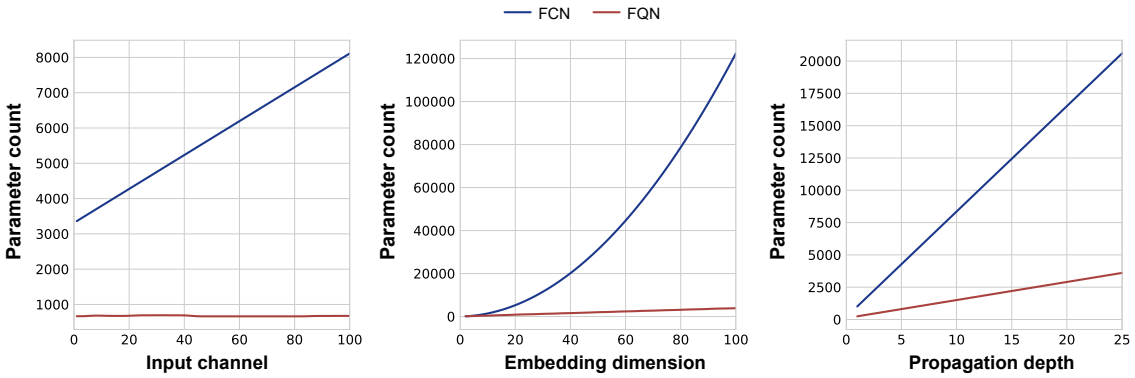

Figure 2: Variation of model parameter count against hyperparameter change. The base hyperparameters are: $C_{in} = 3$, $k_e = 3$, $s_e = 2$, $dim_e = 16$, $k_{prop} = 3$, and $depth = 4$. The parameter change is then observed by varying one hyperparameter at a time. For illustration, we used a Gaussian filter for line smoothing.

alongside previous SOTAs, including TimesNet [50], OS-CNN [48], ROCKET [12], and InceptionTime [20].

3.2 Time Series Classification

According to the performance results outlined in Table 2, FQN outperforms FCN despite having substantially fewer parameters (refer to Table 6). While FQN and FCN outperformed other models in four cases (resulting in a tie), FQN was better overall and surpassed FCN in ten cases. Since our goal with FQN was to develop a model comparable or equivalent to FCN, the results in Table 2 highlight the potential of quantum-based representation learning. The advantage of FQN over FCN comes from its efficient parameter management and self-regularization, which we discuss in the next section.

FQN consistently outperforms the other quantum models across all datasets. In terms of stability, QuanvNet and ESQN exhibit the highest variance in accuracy. Their subpar performance stems from two main limitations: (i) a fixed, small kernel length that restricts their ability to capture temporal relationships effectively and (ii) reliance on a single circuit as a filter, which limits their capacity to extract diverse patterns. Furthermore, ESQN does not have a trainable quanvolution layer, and its randomness significantly impairs learning. Since both models can only handle univariate time series data, we could not evaluate them on all 20 datasets.

Although it is a common notion that more parameters lead to better performance, a model with sufficient parameters to fit the data properly–provided it is well-suited to the task–should not underperform compared to a model with significantly more parameters [2]. This parametric efficiency was evident with FQN, which performed on par with OS-CNN, ROCKET, and InceptionTime—and even surpassing them in some cases—despite being very lightweight (refer to Table 6). FQN also significantly outperformed the current SOTA in time series classification, ModernTCN.

3.3 Parameter Efficiency

Despite both models sharing the same architecture and hyperparameters, FQN is, on average, 6.5 times lighter than FCN (refer to Table 6). Figure 2 illustrates the parameter differences between the two models across various hyperparameter settings. As seen in the graph, the primary difference comes from dim_e, where FCN's parameter count grows exponentially, while FQN shows only a gradual linear increase. While both models experience linear scaling in parameter count as $depth$ increases, the rate of increase is significantly higher for FCN. Additionally, Quanv1D's efficient encoding and decoding processes help keep FQN's parameter count nearly constant for increasing C_{in}, in contrast to FCN's steady linear growth. A higher C_{in} typically requires more qubits (and hence, more parameters), but Quanv1D minimizes this by utilizing all available qubits in its circuits, thereby reducing the total number of filters needed. This synergistic approach makes Quanv1D (and FQN) more efficient in managing parameters.

Table 3 shows that FQN consistently maintains comparable losses across the training, validation, and test sets. This uniformity contrasts with its classical counterpart, FCN, which often overfits and achieves almost zero training loss but much higher validation and test losses. However, after closely looking at Table 3 and how FQN behaves during training, we hypothesize that having more parameters is not the sole reason why FCN fell behind FQN in multiple datasets. To test this hypothesis, we rerun the experiments, adjusting the hyperparameters of FCN to ensure that the trainable parameter for both FCN and FQN is equal. While reducing parameters lessened overfitting in six cases (dropping from 12 to six), the test accuracy declined for ten cases, causing overall performance degradation. Therefore, FQN offers more stable learning than FCN despite the number of parameters used.

3.4 Self-Regularization

Table 3 provides empirical evidence of FQN's self-regularization, which enabled it to generalize more effectively in most test cases. Although this implicit regularization is partly due to the reduced number of parameters, we believe it stems from the fundamental nature of quantum operations within the Quanv1D layer. To illustrate this, consider a single-wire quanvolutional kernel (or quantum circuit) with a single unitary operator. The derivatives of the output, Q_{out}, with respect to θ and λ for a given patch are expressed as follows:

Table 3: The mean loss values of FQN and FCN across datasets with different splits. For FCN*, the total number of trainable parameters matches that of FQN.

	FQN			FCN				FCN*				FCN-FCN*
Dataset	Train	Val	Test	Train	Val	Test	Overfitting?	Train	Val	Test	Overfitting?	Change in accuracy (%)
D1	0.27	0.29	0.31	0.00	0.01	0.23	✓	0.00	0.01	0.17	✓	0.5
D2	0.61	0.68	0.68	0.01	2.87	2.51	✓	0.03	3.21	2.96	✓	0.5
D3	0.80	0.83	0.82	0.00	0.06	0.01	×	0.05	0.08	0.04	×	-0.2
D4†	0.44	0.56	0.50	0.01	0.91	0.87	✓	0.48	0.53	0.51	×	-5.2
D5	0.43	0.60	0.55	0.00	1.29	1.36	✓	0.01	1.09	0.59	✓	6.5
D6	0.40	0.44	0.43	0.00	0.00	0.02	×	0.00	0.02	0.02	×	0.0
D7	0.38	0.39	0.41	0.02	0.26	0.29	✓	0.13	0.48	0.27	✓	1.2
D8	0.68	0.54	0.59	0.17	0.00	0.00	×	0.17	0.08	0.08	×	-2.2
D9†	0.48	0.82	0.57	0.00	4.19	1.70	✓	0.00	3.10	1.38	✓	0.7
D10	0.66	0.64	0.68	0.02	0.01	0.01	×	0.13	0.08	0.07	×	0.0
D11	0.36	0.38	0.38	0.00	0.36	0.17	✓	0.03	0.14	0.09	×	-5.1
D12†	0.57	0.57	0.56	0.02	0.07	0.03	×	0.44	0.68	0.34	×	-2.4
D13	0.58	0.67	0.58	0.01	2.14	1.19	✓	0.55	0.72	0.57	×	-3.2
D14	0.50	0.53	0.51	0.00	0.07	0.03	×	0.01	0.08	0.07	×	-2.2
D15	1.36	1.41	1.38	0.46	1.88	2.35	✓	0.96	1.52	1.67	✓	10.1
D16	0.15	0.19	0.17	0.00	0.01	0.02	×	0.08	0.21	0.21	×	-3.5
D17	0.61	0.71	0.70	0.41	1.07	1.09	✓	0.64	0.75	0.71	×	7.6
D18	1.34	1.43	1.36	0.66	1.75	1.75	✓	1.36	1.40	1.36	×	2.6
D19	0.55	0.75	0.69	0.07	1.85	1.74	✓	0.61	0.85	0.75	×	-2.9
D20†	0.67	0.71	0.70	0.39	1.12	1.14	×	0.68	0.70	0.71	×	-3.7
Overall												-0.1

†For these datasets, the grid search could not identify hyperparameters for FCN* that perfectly matched the parameter count of FQN.
As a result, FCN* has one additional trainable parameter for these cases–for instance, 3401 instead of the exact 3400 for D4.

$$\frac{dQ_{out}}{d\theta} = -\pi(x_1^2 - x_2^2)\sin(\pi\theta) - 2\pi x_1 x_2 \cos(\lambda)\cos(\theta\pi) \quad (9)$$

$$\frac{dQ_{out}}{d\lambda} = 2x_1 x_2 \sin(\lambda)\sin(\theta\pi) \quad (10)$$

Both equations demonstrate a sinusoidal relationship between the derivatives and the weight values: θ and λ. A sinusoidal function, naturally bound between -1 and 1, inherently constrains the gradient update due to its periodicity. Additionally, the amplitude embedding restricts the input values to a range between 0 and 1. As the inputs are the coefficients of the sinusoids, the input scaling limits the magnitude of the gradient updates even further. Together, these factors help prevent extreme gradient values, promoting self-regularization.

Similarly, we can extend this illustration to a two-qubit, two-unitary circuit. The gradients for each wire with respect to the weights are as follows:

$$\frac{dQ_{out}}{d\theta_1} = -\pi(x_1^2 + x_2^2 - x_3^2 - x_4^2)\sin(\theta_1\pi) - 2\pi(x_1x_3 + x_2x_4)\cos(\lambda_1)\cos(\pi\theta_1) \quad (11)$$

$$\frac{dQ_{out}}{d\theta_2} = -\pi(x_1^2 - x_2^2 + x_3^2 - x_4^2)\sin(\theta_2\pi) - 2\pi(x_1x_2 + x_3x_4)\cos(\lambda_2)\cos(\pi\theta_2) \quad (12)$$

$$\frac{dQ_{out}}{d\lambda_1} = (x_1x_3\sin(\lambda_1) + x_2x_4\sin(\lambda_1))\sin(\pi\theta_1) \quad (13)$$

$$\frac{dQ_{out}}{d\lambda_2} = (x_1x_2\sin(\lambda_2) + x_3x_4\sin(\lambda_2))\sin(\pi\theta_2) \quad (14)$$

θ_i and λ_i represent the weights of the i-th wire. The regularization effect is also evident in Equations (11, 12, 13, and 14). The gradient update value discussed here applies to a single layer, but because the model is fully quanvolutional, adding more layers will not affect this behavior. Since gradient updates occur via a multiplicative chain rule, a similar regularization effect will propagate across all layers. The derivation for the single-wire circuit can be found in Appendix F, and the equations for the two-wire circuit were derived using SymPy [31]. Note that the derivatives with respect to ϕ are always zero.

3.5 Impact of Finite Shots

Despite the promise of FQN to classify real-world data, its implementation on actual quantum computers remains a challenge. Quanv1D is primarily theoretical, as we rely on analytical or raw expectation values. In practice, this approach is not feasible because expectation values must be measured using a finite number of shots. To assess our model's performance under such conditions, we conducted experiments starting with 1000 shots and continued until FQN achieved the analytical benchmark.

Due to the high computational cost of simulating individual measurement shots, we approximated their effects as sampling errors. Specifically, we introduced stochastic perturbations to the probability amplitudes to model the uncertainty from finite-shot measurements. These perturbations were drawn from a normal

Table 4: FQN's performance under statistical noise. The model is evaluated five times for each shot count on the test sets. We present the mean and standard deviation for test accuracy (%).

	Number of shots					
Dataset	1000	10000	100000	1000000	10000000	Analytical
D1	95.1 (0.8)	96.4 (0.8)	–	–	–	96.4 (1.2)
D2	56.3 (0.3)	60.3 (0.7)	–	–	–	59.5 (1.0)
D3	96.8 (0.4)	99.0 (0.4)	–	–	–	99.0 (0.4)
D4	72.7 (0.9)	75.6 (0.6)	77.1 (0.4)	78.2 (0.2)	78.8 (0.1)	78.7 (1.0)
D5	80.5 (2.1)	–	–	–	–	80.5 (4.1)
D6	93.9 (0.8)	–	–	–	–	93.3 (1.2)
D7	95.2 (1.7)	95.8 (1.7)	–	–	–	95.8 (1.7)
D8	97.5 (1.4)	98.1 (0.7)	–	–	–	98.1 (1.3)
D9	76.1 (1.1)	–	–	–	–	75.7 (2.3)
D10	32.3 (1.1)	53.5 (2.1)	71.0 (0.0)	100.0 (0.0)	–	100.0 (0.0)
D11	96.2 (0.1)	96.9 (0.1)	–	–	–	96.9 (0.3)
D12	85.4 (0.0)	99.5 (1.1)	–	–	–	99.5 (1.1)
D13	72.9 (1.8)	–	–	–	–	72.7 (1.1)
D14	96.0 (0.8)	98.2 (1.3)	100.0 (0.0)	–	–	99.3 (1.0)
D15	30.9 (0.5)	–	–	–	–	30.9 (2.1)
D16	98.4 (0.5)	99.3 (0.3)	–	–	–	99.2 (0.3)
D17	57.4 (3.0)	–	–	–	–	56.8 (2.4)
D18	26.4 (1.9)	28.5 (1.2)	32.8 (1.2)	34.5 (1.0)	35.7 (1.0)	35.7 (3.5)
D19	54.7 (1.6)	56.2 (0.7)	–	–	–	56.1 (2.8)
D20	52.4 (1.4)	–	–	–	–	52.4 (1.1)

distribution, $\mathcal{N}(0, \frac{p(1-p)}{n})$, where the variance corresponds to the squared standard error of a Bernoulli process over n shots. Since each shot outcome follows a Bernoulli distribution, this formulation approximately captures the statistical fluctuations. With this method, we significantly reduced computational time and memory usage compared to explicitly simulating each shot or employing parallel processing, allowing us to simulate any number of shots efficiently. The results of the shot-based analysis are presented in Table 4.

Since we begin with a sufficiently large number of shots (1000), FQN achieves analytical accuracy in the first run for some cases. However, a consistent trend is that FQN converges as the number of shots increases, aligning with the law of large numbers [15]. Interestingly, the introduced noise can sometimes enhance performance, even surpassing the noise-free analytical baseline, as observed across multiple datasets. This result is in accordance with previous studies on QML [9, 13, 14] and classical neural networks [4, 38], which suggest that noise can, in certain cases, enhance data learning, promote robustness, and improve generalizability.

3.6 Ablation Study

Although FQN primarily uses a quantum layer, Quanv1D, the model itself is a classical-quantum hybrid, as we had to use batch normalization and ReLU activation in our implementation. We have used the batch normalization layers to help maintain an even distribution over the limited working range of [-1, 1] of Quanv1D by mitigating the covariance shift and ReLU activations to add nonlinearity in the otherwise linear model to facilitate better training. We found that removing batch normalization layers reduces the accuracy of the model by 32%, while removing ReLU decreases performance by 6% on average, highlighting the importance of their functionalities. In particular, batch normalization plays a critical role in stabilizing the outputs, which otherwise exhibit low variance or skewed/biased means. Without batch normalization layers, the model simply fails to converge.

4 Concluding Remarks

The primary objective of this study was to introduce a QML algorithm for quantum computers, aiming to create a quantum equivalent of a well-established and widely used classical method, 1D convolution. We designed our proposed quanvolution algorithm, an analog to convolution, with NISQ-related constraints in mind, and it proved effective in temporal data learning tasks. In fact, it was able to outperform contemporary classical models in some cases, thanks to its efficient parameter management and inherent regularization. Despite our model being a theoretical framework, it remained effective when we simulated a realistic scenario with statistical noise. Our proposed model, FQN, incorporates nonlinear activation functions, which present a challenge in a quantum environment. But the computational overhead for these activation functions is negligible, and the network should, in theory, work well in a classical-quantum hybrid framework. Previous studies have demonstrated promising results for similar hybrid approaches with smaller models [45, 46]. As the world moves toward fault-tolerant quantum computing with increased qubits, we are optimistic that, under the right conditions, our model will achieve performance close to its theoretical potential. Nonetheless, much progress is still required before reaching that point.

4.1 Limitations

For this study, we tested our FQN only on classification tasks and have yet to explore regression and data imputation problems. Using a fully quanvolutional structure for these tasks will be challenging, as Quanv1D's output is in the range of -1 to 1, which limits its applicability for tasks that require output values beyond this range. Additionally, for multi-class problems, we noticed that training FQN gets complicated due to the limited number of parameters and/or the usage of classical optimizers on a quantum model.

In addition to these limitations, we believe it is important to address the constraints related to hardware implementation. Our quanvolutional layer is mainly theoretical and is simulated using classical computers. Given the constraints associated with the NISQ era, it remains uncertain whether we can achieve this level of implementation, as we did not have access to real hardware to verify our algorithm. Moreover, although amplitude encoding offers improved parameter efficiency for the model, it significantly increases circuit depth [22, 41], posing further challenges. Finally, the high computational complexity of FQN makes classical simulation difficult. For comparison, the time complexity of quanvolution is $O(k^2 \times c_{in}^2 \times c_{out})$, whereas that of convolution is $O(k \times c_{in} \times c_{out})$, where k is the kernel length, c_{in} is the input dimension, and c_{out} is the number of output channels. Optimizing the design to reduce simulation complexity remains an area for future exploration.

4.2 Future Work

Our immediate priority is to improve the model's performance with a larger number of classes and streamline its design to minimize time complexity. After completing the classification phase, we will address the output range limitation of -1 to 1 for regression and forecasting tasks. While the output of the quantum layer cannot be altered, we aim to design an activation function that maps outputs to a higher range while preserving distinct quantum properties. Evaluating the model on diverse temporal data learning tasks will help establish the true operational scope of 1D quanvolution. Additionally, we plan to integrate quantum-aware optimizers to enable more informed gradient updates, promoting stable and robust learning. Finally, considering the aforementioned constraints, transitioning from 1D to 2D quanvolution while maintaining this level of modularity will be a significant challenge. Nevertheless, we believe that this progression is the logical next step, especially for expanding applications into image processing and computer vision.

References

[1] Google Quantum AI and Collaborators. 2024. Quantum Error Correction Below the Surface Code Threshold. *Nature* 638, 8052 (2024), 920.

[2] Constantin Aliferis and Gyorgy Simon. 2024. Overfitting, Underfitting and General Model Overconfidence and Under-Performance Pitfalls and Best Practices in Machine Learning and AI. In *Artificial Intelligence and Machine Learning in Health Care and Medical Sciences: Best Practices and Pitfalls*. Springer, , 477–524.

[3] Anthony Bagnall et al. 2018. The UEA Multivariate Time Series Classification Archive, 2018. arXiv:1811.00075 Retrieved from https://arxiv.org/abs/1811.00075.

[4] Christopher M Bishop. 1995. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, England.

[5] Katherine S Button et al. 2013. Power Failure: Why Small Sample Size Undermines the Reliability of Neuroscience. *Nat. Rev. Neurosci.* 14, 5 (2013), 365–376.

[6] Davide Castelvecchi. 2023. IBM Releases First-Ever 1,000-Qubit Quantum Chip. *Nature* 624, 7991 (2023), 238–238.

[7] Kumar Chellapilla, Sidd Puri, and Patrice Simard. 2006. High Performance Convolutional Neural Networks for Document Processing. In *Tenth International Workshop on Frontiers in Handwriting Recognition*, Guy Lorette (Ed.). Université de Rennes 1, Suvisoft, La Baule (France).

[8] Iris Cong, Soonwon Choi, and Mikhail D Lukin. 2019. Quantum Convolutional Neural Networks. *Nat. Phys.* 15, 12 (2019), 1273–1278.

[9] Andrew W Cross, Graeme Smith, and John A Smolin. 2015. Quantum Learning Robust Against Noise. *Phys. Rev. A* 92, 1 (2015), 012327.

[10] Hoang Anh Dau et al. 2019. The UCR Time Series Archive. *IEEE/CAA J. Automatica Sinica* 6, 6 (2019), 1293–1305.

[11] Saeideh Davoudi, Tyler Schwartz, Aurélie Labbe, Laurel Trainor, and Sarah Lippé. 2023. Inter-Individual Variability During Neurodevelopment: An Investigation of Linear and Nonlinear Resting-State EEG Features in an Age-Homogenous Group of Infants. *Cereb. Cortex* 33, 13 (2023), 8734–8747.

[12] Angus Dempster, François Petitjean, and Geoffrey I Webb. 2020. ROCKET: Exceptionally Fast and Accurate Time Series Classification Using Random Convolutional Kernels. *Data Min. Knowl. Discov.* 34, 5 (2020), 1454–1495.

[13] Laia Domingo, G Carlo, and F Borondo. 2023. Taking Advantage of Noise in Quantum Reservoir Computing. *Sci. Rep.* 13, 1 (2023), 8790.

[14] Yuxuan Du, Min-Hsiu Hsieh, Tongliang Liu, Dacheng Tao, and Nana Liu. 2021. Quantum Noise Protects Quantum Classifiers Against Adversaries. *Phys. Rev. Res.* 3, 2 (2021), 023153.

[15] Michael J Evans and Jeffrey S Rosenthal. 2004. *Probability and Statistics: The science of Uncertainty*. Macmillan, United Kingdom.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, Las Vegas, USA, 770–778.

[17] Maxwell Henderson, Samriddhi Shakya, Shashindra Pradhan, and Tristan Cook. 2020. Quanvolutional Neural Networks: Powering Image Recognition with Quantum Circuits. *Quantum Mach. Intell.* 2, 1 (2020), 2.

[18] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely Connected Convolutional Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, Honolulu, USA, 4700–4708.

[19] Tak Hur, Leeseok Kim, and Daniel K Park. 2022. Quantum Convolutional Neural Network for Classical Data Classification. *Quantum Mach. Intell.* 4, 1 (2022), 3.

[20] Hassan Ismail Fawaz et al. 2020. InceptionTime: Finding AlexNet for Time Series Classification. *Data Min. Knowl. Discov.* 34, 6 (2020), 1936–1962.

[21] Yu Jing et al. 2022. RGB Image Classification with Quantum Convolutional Ansatz. *Quantum Inf. Process.* 21, 3 (2022), 101.

[22] Ruba Kharsa, Ahmed Bouridane, and Abbes Amira. 2023. Advances in Quantum Machine Learning and Deep learning for Image Classification: A Survey. *Neurocomputing* 560 (2023), 126843.

[23] Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 Retrieved from https://arxiv.org/abs/1412.6980.

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger (Eds.), Vol. 25. Curran Associates, Inc., Lake Tahoe, USA.

[25] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-Based Learning Applied to Document Recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[26] Yue Li et al. 2024. Detection and Identification of Power Quality Disturbance Signals in New Power System Based on Quantum Classic Hybrid Convolutional Neural Networks. In *International Conference on Data Security and Privacy Protection*. Springer, Xi'an, China, 187–203.

[27] Zhaokai Li, Xiaomei Liu, Nanyang Xu, and Jiangfeng Du. 2015. Experimental Realization of a Quantum Support Vector Machine. *Phys. Rev. Lett.* 114, 14 (2015), 140504.

[28] Donghao Luo and Xue Wang. 2024. ModernTCN: A Modern Pure Convolution Structure for General Time Series Analysis. In *The Twelfth International Conference on Learning Representations*. , Vienna, Austria, 1–43.

[29] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. 2016. Understanding the Effective Receptive Field in Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (Eds.), Vol. 29. Curran Associates, Inc., Barcelona, Spain.

[30] Denny Mattern, Darya Martyniuk, Henri Willems, Fabian Bergmann, and Adrian Paschke. 2021. Variational Quanvolutional Neural Networks with Enhanced Image Encoding. arXiv:2106.07327 Retrieved from https://arxiv.org/abs/2106.07327.

[31] Aaron Meurer et al. 2017. SymPy: Symbolic Computing in Python. *PeerJ Comput. Sci.* 3 (2017), e103.

[32] Michael A Nielsen and Isaac L Chuang. 2010. *Quantum Computation and Quantum Information*. Cambridge University Press, .

[33] Adam Paszke et al. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc., Vancouver, Canada.

[34] David Peral-García, Juan Cruz-Benito, and Francisco José García-Peñalvo. 2024. Systematic Literature Review: Quantum Machine Learning and Its Applications. *Comput. Sci. Rev.* 51 (2024), 100619.

[35] Frank Phillipson, Niels Neumann, and Robert Wezeman. 2023. Classification of Hybrid Quantum-Classical Computing. In *International Conference on Computational Science*. Springer, Prague, Czech Republic, 18–33.
[36] Sharanya Prabhu, Shourya Gupta, Gautham Manuru Prabhu, Aarushi Vishal Dhanuka, and K Vivekananda Bhat. 2023. QuCardio: Application of Quantum Machine Learning for Detection of Cardiovascular Diseases. *IEEE Access* 11 (2023), 136122–136135.
[37] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. 2014. Quantum Support Vector Machine for Big Data Classification. *Phys. Rev. Lett.* 113, 13 (2014), 130503.
[38] Russell Reed and Robert J MarksII. 1999. *Neural Smithing*. Bradford Books, Cambridge, MA.
[39] Mayra Alejandra Rivera-Ruiz, Sandra Leticia Juárez-Osorio, Andres Mendez-Vazquez, José Mauricio López-Romero, and Eduardo Rodriguez-Tello. 2023. 1D Quantum Convolutional Neural Network for Time Series Forecasting and Classification. In *Mexican International Conference on Artificial Intelligence*. Springer, Yucatán, Mexico, 17–35.
[40] Aansh Savla, Ali Abbas Kanadia, Deep Mehta, and Kriti Srivastava. 2022. GQNN: Greedy Quanvolutional Neural Network Model. In *International Conference on Image Processing and Capsule Networks*. Springer, Bangkok, Thailand, 397–410.
[41] Maria Schuld and Francesco Petruccione. 2018. *Supervised Learning with Quantum Computers*. Springer, .
[42] Rebh Soltani, Emna Benmohamed, and Hela Ltifi. 2024. Hybrid Quanvolutional Echo State Network for Time Series Prediction. In *Proceedings of the 16th International Conference on Agents and Artificial Intelligence (ICAART 2024)*, Vol. 2. SCITEPRESS, Rome, Italy, 40–46.
[43] S Sridevi, T Kanimozhi, K Issac, and M Sudha. 2022. Quanvolution Neural Network to Recognize Arrhythmia from 2D Scalogram Features of ECG Signals. In *2022 International Conference on Innovative Trends in Information Technology (ICITIIT)*. IEEE, Kottayam, India, 1–5.
[44] Christian Szegedy et al. 2015. Going Deeper with Convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, Boston, USA, 1–9.
[45] Francesco Tacchino, Panagiotis Barkoutsos, Chiara Macchiavello, Ivano Tavernelli, Dario Gerace, and Daniele Bajoni. 2020. Quantum Implementation of an Artificial Feed-Forward Neural Network. *Quantum Sci. Technol.* 5, 4 (2020), 044010.
[46] Francesco Tacchino, Chiara Macchiavello, Dario Gerace, and Daniele Bajoni. 2019. An Artificial Neuron Implemented on an Actual Quantum Processor. *npj Quantum Inf.* 5, 1 (2019), 26.
[47] Mingxing Tan and Quoc Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, California, USA, 6105–6114.
[48] Wensi Tang, Guodong Long, Lu Liu, Tianyi Zhou, Michael Blumenstein, and Jing Jiang. 2022. Omni-Scale CNNs: A Simple and Effective Kernel Size Configuration for Time Series Classification. In *The Tenth International Conference on Learning Representations*. , Virtual, 1–17.
[49] Ubaid Ullah and Begonya Garcia-Zapirain. 2024. Quantum Machine Learning Revolution in Healthcare: A Systematic Review of emerging perspectives and applications. *IEEE Access* 12 (2024), 11423–11450.
[50] Haixu Wu, Tengge Hu, Yong Liu, Hang Zhou, Jianmin Wang, and Mingsheng Long. 2023. TimesNet: Temporal 2D-Variation Modeling for General Time Series Analysis. In *The Eleventh International Conference on Learning Representations*. , Kigali, Rwanda, 1–23.
[51] Chao-Han Huck Yang et al. 2021. Decentralizing Feature Extraction with Quantum Convolutional Neural Network for Automatic Speech Recognition. In *2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, Toronto, Canada, 6523–6527.
[52] Fisher Yu and Vladlen Koltun. 2016. Multi-Scale Context Aggregation by Dilated Convolutions. arXiv:1511.07122 Retrieved from https://arxiv.org/abs/1511.07122.
[53] Juping Zhang, Gan Zheng, Toshiaki Koike-Akino, Kai-Kit Wong, and Fraser A. Burton. 2024. Hybrid Quantum-Classical Neural Networks for Downlink Beamforming Optimization. *IEEE Trans. Wirel. Commun.* 23, 11 (2024), 16498–16512.

A Preliminaries

Quantum bits, or qubits, are the fundamental units of quantum computation. Unlike classical bits, which can only exist in one of two definite states (0 or 1), qubits can be in the 0 state ($|0\rangle$), the 1 state ($|1\rangle$), or any linear combination (superposition) of these states, expressed as $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$. Here, α and β are complex numbers that satisfy the condition $|\alpha|^2+|\beta|^2 = 1$. This superposition property allows quantum operations to act on multiple states simultaneously, which classical computers cannot achieve. When combined with the phenomena of entanglement and interference, superposition forms the basis of "quantum advantage"–the ability of quantum systems to solve specific problems more efficiently than classical systems.

Quantum circuits provide the framework for modeling quantum computations; wires represent qubits, and gates correspond to quantum operations that manipulate these qubits. In the context of applying QML to classical data, quantum circuits typically consist of three main phases: encoding, manipulation, and measurement.

(1) **Encoding:** In this phase, classical information is mapped into a quantum Hilbert space, preparing the quantum system for computation. Two widely used encoding methods are angle encoding and amplitude encoding. Angle encoding represents classical values as angles of rotation gates applied to qubits, typically starting from the $|0\rangle$ state. This method requires n qubits to encode n classical features. Amplitude encoding or embedding, in contrast, stores classical data in the amplitudes of a quantum state, allowing n qubits to represent up to 2^n classical values. While amplitude encoding is highly efficient in terms of qubit usage, preparing such states can be computationally intensive.
(2) **Manipulation:** After encoding, the quantum data is processed using a sequence of quantum gates. Researchers often design parameterized quantum circuits, known as ansatz, which include gates such as Hadamard, Controlled-NOT (CNOT), and rotations around the X, Y, and Z axes. These gates enable critical quantum phenomena for quantum advantage. For example, the Hadamard gate creates superposition, while the CNOT gate establishes entanglement.
(3) **Measurement:** The final stage involves measuring quantum states to extract classical outcomes. Measurement causes the quantum state to collapse into one of its basis states (either 0 or 1). Due to the probabilistic nature of quantum measurement, multiple repetitions (referred to as "shots") are performed to gather sufficient statistics. With enough measurements or experimental reruns, the outcomes reliably reflect the expected or analytical results of the quantum computation.

B Experimental Setup

In this study, we conducted the experiments on an NVIDIA GeForce RTX 4060 Ti 16GB GPU using PyTorch [33].

C Dataset Setup

The datasets used in this experiment were taken from the UEA and UCR archives [3, 10]. Among the 150+ datasets available in the archive, we have considered a subset of 20. The selected datasets, along with their descriptions, are provided in Table 1. Our dataset selection criteria were primarily based on the domain and its real-life applicability, i.e., we wanted to cover as many domains as possible to test our proposed model. The considered datasets come from 15 different domains and include a mix of both binary and multi-class classification tasks. A major portion of the selected datasets is univariate, as QuanvNet [39] and ESQN [42], two of our baselines, only work on univariate data. One exception to the criteria was the selection of multiple EEG datasets. This was due to the high

Table 5: Total trainable parameters of FQN with its hyperparameters.

Dataset	Batch size	Parameters	LR decay factor	k_e	s_e	dim_e	k_{prop}	$depth$
D1	64	856	0.85	9	1	16	3	4
D2	256	908	0.85	15	3	16	3	3
D3	128	2988	0.85	15	2	48	3	4
D4	256	2576	0.75	21	2	32	3	4
D5	32	1808	0.75	31	4	16	3	4
D6	64	4434	0.75	33	2	32	7	4
D7	32	3200	0.75	31	5	24	5	5
D8	32	6098	0.8	45	4	32	9	4
D9	64	3400	0.9	21	5	32	5	5
D10	32	2796	0.75	33	5	16	9	5
D11	128	3400	0.8	21	8	32	5	5
D12	32	3440	0.85	41	10	16	7	6
D13	32	1166	0.9	15	20	8	7	5
D14	64	6072	0.85	25	3	64	3	5
D15	128	4858	0.85	45	10	24	7	7
D16	128	3618	0.85	41	5	24	5	5
D17	64	4882	0.85	41	9	32	5	5
D18	128	1488	0.75	9	2	8	5	9
D19	64	728	0.75	5	2	16	3	4
D20	16	524	0.9	3	25	16	3	3

Table 6: Parameter counts across datasets. The table shows the ratio of total trainable parameters for each model relative to FQN. Bold indicates the least number of parameters.

Dataset	FQN	FCN	QuanvNet*	ESQN*	OS-CNN	ROCKET	InceptionTime	ModernTCN	TimesNet†
D1	×**1.0**	×4.0	×108.0	×1.2	×567.4	×23.4	×552.2	×11.8	×203.6
D2	×**1.0**	×3.0	×101.8	×1.1	×377.3	×22.0	×520.5	×13.7	×194.5
D3	×1.0	×9.1	×23.4	×**0.8**	×85.3	×29.8	×117.4	×5.0	×45.7
D4	×1.0	×7.9	×27.2	×**0.3**	×91.3	×5.9	×139.0	×4.0	×52.3
D5	×1.0	×2.1	×51.1	×**0.6**	×161.1	×11.1	×261.4	×8.1	×99.0
D6	×1.0	×6.8	×20.8	×**0.2**	×61.4	×4.5	×106.6	×4.0	×41.0
D7	×1.0	×4.9	×28.9	×**0.3**	×71.6	×6.3	×147.7	×9.5	×60.9
D8	×1.0	×6.4	×15.2	×**0.3**	×37.6	×13.1	×77.6	×8.7	×35.6
D9	×1.0	×7.9	×27.2	×**0.3**	×69.3	×5.9	×139.0	×12.2	×60.5
D10	×1.0	×4.4	×33.2	×**0.5**	×84.3	×21.4	×169.1	×23.7	×82.4
D11	×1.0	×6.4	×23.3	×**0.4**	×59.0	×15.0	×118.4	×26.8	×67.9
D12	×1.0	×3.4	×27.0	×**0.4**	×68.5	×17.4	×137.4	×44.4	×92.1
D13	×1.0	×2.2	×79.3	×**0.9**	×202.2	×17.2	×405.4	×1024.4	–
D14	×**1.0**	×11.1	–	–	×54.2	×13.2	×78.0	×17.3	×32.8
D15	×**1.0**	×6.1	–	–	×53.6	×18.0	×106.7	×157.5	×89.6
D16	×**1.0**	×4.0	–	–	×80.9	×6.8	×160.4	×56.2	×69.7
D17	×**1.0**	×7.2	–	–	×49.5	×4.1	×97.1	×118.8	×50.6
D18	×**1.0**	×2.9	–	–	×207.9	×68.0	×403.6	×516.1	×191.2
D19	×**1.0**	×7.6	–	–	×7195.1	×27.5	×656.4	×576.4	×245.3
D20	×**1.0**	×10.6	–	–	×565.8	×38.2	×925.1	×25452.0	×707.7
Average	×**1.0**	×6.5	–	–	×174.3	×14.3	×157.2	×292.7	–

*The results for QuanvNet and ESQN are incomplete because these models can handle univariate data only.

†Due to its embedding scheme, TimesNet cannot handle sequence lengths larger than 5000. As such, TimesNet's D13 performance is incomplete.

variability and low statistical power of EEG datasets [5, 11], along with their high dimensionality.

All datasets were processed following the same procedure. We first applied Z-normalization to the entire dataset, followed by a 60:20:20 split for training, validation, and testing. For multi-dimensional data, we performed Z-normalization on each input channel individually. We saved the normalization factors and data splits to ensure reproducibility and fair comparisons among models trained on the same datasets. Additionally, we calculated class weights and integrated them into the cross-entropy loss function to address class imbalances in several datasets.

D Training Setup

We trained each model for 200 epochs, opting for a learning rate scheduler instead of early stopping. As mentioned before, our objective was to evaluate the differences in learning between Quanv1D and traditional convolutional layers. The scheduler had a patience of 5 epochs and a relative threshold of 0.001, monitoring the validation loss to reduce the learning rate by different factors when performance plateaued. We varied the decay factor to ensure stable training and minimize overfitting. However, for consistency, the decay factor remained the same across all comparison models for each dataset. The data specific factors are presented in Table 5.

We optimized all the models using Adam [23], starting with default values set by PyTorch. The initial learning rate for all datasets was 0.01, except for D10 and D12, where it was 0.001. We repeated all the experiments five times without using any specific seeds.

E Model Setup

Table 5 presents FQN's hyperparameters for each dataset. Note that, for fairness, we also use the same batch size for other models under comparison. In the case of FCN, the hyperparameters are identical to the values presented in Table 5. In the table, *depth* refers to the number of propagation layers.

We utilized the official implementations retaining the hyperparameters recommended by respective authors for the SOTA convolutional models. Since no code was available, we implemented the quantum models from scratch based on the approach outlined in the respective papers.

F Gradient Derivation

Let us consider a one-wire, one-unitary quantum circuit where the number of inputs is 2. Before being processed by the circuit, the inputs undergo amplitude embedding, such that the input vector is given by:

$$|x_{input}\rangle = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \tag{15}$$

The output state of the quantum circuit after applying a unitary transformation parameterized by angles θ, ϕ, and λ is described as:

$$\begin{aligned} |\psi_o\rangle &= U(\theta, \phi, \lambda) * x_{input} \\ &= \begin{pmatrix} x_1 * \cos(\theta\frac{\pi}{2}) - x_2 * e^{i\lambda} \sin(\theta\frac{\pi}{2}) \\ x_1 * e^{i\phi} \sin(\theta\frac{\pi}{2}) + x_2 * e^{i(\phi+\lambda)} \cos(\theta\frac{\pi}{2}) \end{pmatrix} \end{aligned} \tag{16}$$

After passing through the measurement block, the output Q_{out} is computed as the expectation value of the Pauli-Z operator:

$$Q_{out} = \langle\psi_o|Z|\psi_o\rangle \tag{17}$$

Substituting $|\psi_o\rangle$ into this expression and simplifying, we obtain:

$$\begin{aligned} Q_{out} &= x_1^2 \cos^2(\theta\frac{\pi}{2}) + x_2^2 \sin^2(\theta\frac{\pi}{2}) - x_1 x_2 \cos\lambda \sin(\pi\theta) \\ &\quad -x_1^2 \sin^2(\theta\frac{\pi}{2}) - x_2^2 \cos^2(\theta\frac{\pi}{2}) - x_1 x_2 \cos\lambda \sin(\pi\theta) \\ &= (x_1^2 - x_2^2)\cos^2(\theta\frac{\pi}{2}) - (x_1^2 - x_2^2)\sin^2(\theta\frac{\pi}{2}) - 2x_1 x_2 \cos\lambda \sin(\pi\theta) \end{aligned} \tag{18}$$

Now, the derivatives of Q_{out} with respect to θ, λ, and ϕ are:

$$\begin{aligned} \frac{dQ_{out}}{d\theta} &= -2\frac{\pi}{2}(x_1^2 - x_2^2)\cos(\theta\frac{\pi}{2})\sin(\theta\frac{\pi}{2}) \\ &-2\frac{\pi}{2}(x_1^2 - x_2^2)\sin(\theta\frac{\pi}{2})\cos(\theta\frac{\pi}{2}) - 2\pi x_1 x_2 \cos(\lambda)\sin(\pi\theta) \\ &= -\pi(x_1^2 - x_2^2)\sin(\pi\theta) - 2\pi x_1 x_2 \cos(\lambda) cos(\pi\theta) \end{aligned} \tag{19}$$

$$\frac{dQ_{out}}{d\lambda} = 2x_1 x_2 \sin(\lambda)\sin(\pi\lambda) \tag{20}$$

$$\frac{dQ_{out}}{d\phi} = 0 \tag{21}$$

G Parameter Efficiency (Extended)

Compared to SOTA convolutional models such as InceptionTime, OS-CNN, TimesNet, and ModernTCN, FQN is over 150 times lighter (refer to Table 6). It is about 14 times more parameter-efficient than ROCKET, the widely recognized industry-standard lightweight time series model. The only model with fewer parameters than FQN is ESQN, which achieves this by having no trainable layers aside from a single linear layer.

H FQN vs Quantum Noise

Related to practical deployment, we introduced finite-shot analysis for realistic measurement scenarios. However, considering only measurement error provides an incomplete picture of the noise present in real-world quantum systems. As such, after the review comments, we extended our noise model beyond measurement error by incorporating three complementary mechanisms: amplitude perturbation, coherent unitary misrotations, and depolarizing noise. Amplitude perturbation simulates numerical imprecision and partial decoherence by adding small Gaussian noise ($\epsilon = 0.001$) and dropout ($p = 0.01$) to the state amplitudes, modeling gate infidelity and depth-induced noise. Coherent unitary noise captures analog control errors through systematic gate misrotations, implemented by perturbing rotation angles with Gaussian noise of the same scale. This models calibration drift and hardware inaccuracies. Depolarizing noise is approximated by randomly applying a single-qubit Pauli operator (X, Y, or Z) with 1% probability, introducing stochastic errors similar to environmental decoherence. While this approach does not capture the full statistical mixing of a true depolarizing channel, it serves as a lightweight, differentiable approximation compatible with state-vector simulation. Together, these mechanisms offer a practical and expressive model of NISQ-era noise. We recognize that actual hardware may introduce further complications, but we aimed to simulate its behavior as closely as possible. Even under these noisy conditions, FQN showed robustness (around 3% overall performance degradation), which is still comparable to some SOTA architectures.