IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 14 April 2025, accepted 15 May 2025, date of publication 21 May 2025, date of current version 2 June 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3572416

TUTORIAL

Framework for Implementing Quantum Neural Networks in Wireless Communications

SALIK SIDDIQUI, (Student Member, IEEE), JUSTIN HOLZER, (Student Member, IEEE), JOSHUA MALCARNE, (Student Member, IEEE), GALAHAD M. B. WERNSING, AND ALEXANDER M. WYGLINSKI, (Senior Member, IEEE)

Electrical and Computer Engineering Department, Worcester Polytechnic Institute, Worcester, MA 01609, USA

Corresponding author: Salik Siddiqui (ssiddiqui@wpi.edu)

ABSTRACT While quantum computing is already being employed in different domains to solve large scale and highly complex optimization problems, the wireless communications community has only recently begun exploring the potential of Quantum Machine Learning (QML). This paper provides a reference design approach that allows the wireless community to effectively leverage QML in optimizing their own wireless communications networks. Such an approach will become increasingly valuable as quantum computing matures and conventional computing methods reach their practical limits due to the high dimensionality of optimization problems. We present a detailed methodology enabling wireless communications practitioners to construct their own Quantum Neural Networks (QNN) implementations and benchmark their performance against conventional techniques. Optimization strategies for wireless communication channels using QNNs are explored. The resulting framework provides a generalizable, easy-to-adopt methodology for integrating QML into wireless networks, requiring minimal prior knowledge of quantum computing.

INDEX TERMS Quantum neural network, quantum machine learning, variational quantum circuit, quantum computing, neural network, machine learning, wireless communication, network optimization.

I. INTRODUCTION

As wireless communication networks continue to advance and improve, the demand for increased computational power has become inevitable [1], [2], [3], [4]. This surge is driven by several factors, including the development of cognitive radio technology and the integration of artificial intelligence algorithms [2], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19]. These advances enable networks to autonomously and dynamically adapt to spectrum availability, interference conditions, changing channel conditions, and reliability requirements [20], [21], [22]. Consequently, the processing requirements for real-time decision making and data analysis have increased.

To meet these demands, the potential of quantum computing has emerged as a promising solution [23], [24], [25]. Quantum computing has the potential to address these challenges by leveraging concepts in quantum mechanics to perform computations at speeds that far surpass classical computers. Using quantum computing's significant parallel processing capabilities, future wireless networks can achieve unprecedented levels of efficiency, scalability, and adaptability, thus ushering in a new era of high-performance telecommunications infrastructure [26].

The intersection of quantum computing and wireless communications represents a frontier of technological advancement [27], poised to revolutionize wireless communication networks [13], [14], [15], [16], [17]. Commercial 5G communication networks are complex and require the handling of a large number of devices [28]. Futhermore, 6G and future cellular networks promise additional complexity and even more devices, which could become impossible to manage if employing conventional computing technology. Current research and development are now focused on leveraging quantum computing's capabilities to address critical challenges in wireless networks, such as improvements

The associate editor coordinating the review of this manuscript and approving it for publication was Ayaz Ahmad.

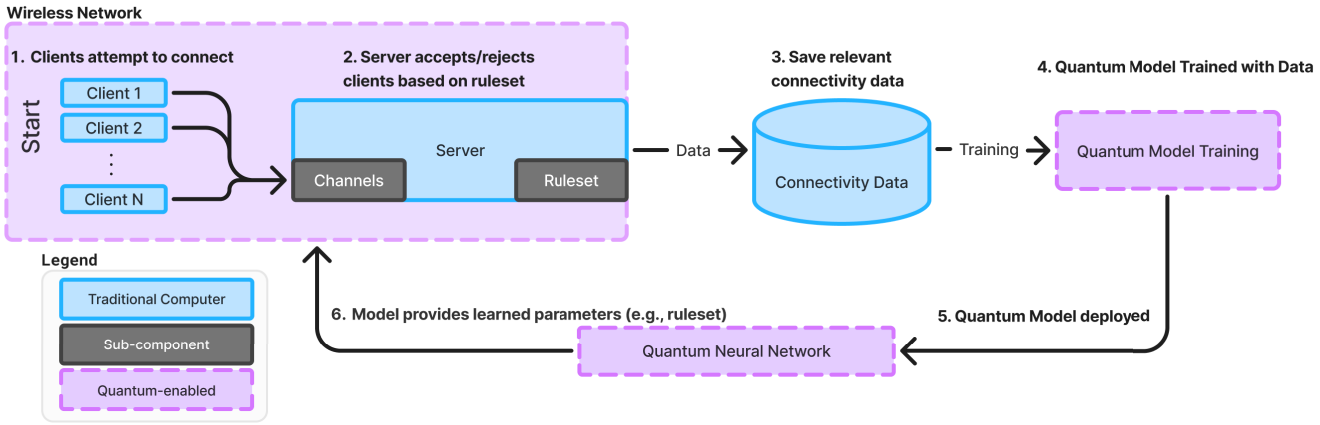

FIGURE 1. High level block diagram of system components. Data is extracted from network, and used to train quantum model. Resulting QNN is used to enable decision-making in network. A detailed explanation is presented in Section IV-A.

in cryptography, transmitter localization, route planning, simulations of complex propagation environments, power allocation, edge computing and Ultra-Reliable Low Latency Communications (URLLC) [13], [15], [16], [17], [19], [29], [30], [31], [32], [33]. Quantum algorithms have shown promise in solving combinatorial optimization problems, which are prevalent in network resource allocation and scheduling tasks [94]. Despite these advancements, practical implementations of quantum computing in wireless communications are still in their infancy and are yet to prove their viability, not to mention their efficacy and reliability [35], [36]. However, with ongoing research and collaboration between academia and industry, quantum computing has the potential to redefine the capabilities and efficiency of future wireless communication networks.

To the authors' knowledge, a detailed framework for the wireless community describing exactly how to implement quantum artificial intelligence algorithms [37] aimed at wireless communications applications is notably absent from the open literature. This paper addresses this knowledge gap and serves as a do-it-yourself (DIY) document for describing the process of performing optimization of wireless communication networks via Quantum Neural Networks (QNNs) (see Figure 1).

In this paper, we present the following novel contributions:

- A generalized architecture for implementing QNNs and Quantum Machine Learning (QML) [37] into a target wireless network.
- An implementation example of how a QNN can be used to solve a wireless communications optimization problem.
- A practical step-by-step tutorial on tuning and evaluating the performance of our example QNN which can be used by those in the wireless community who are interested to apply this framework to their own implementations for other wireless applications.

The framework described in the following sections will provide guidance on how to use QNNs for future applications in wireless communications. The remainder of the paper is organized as follows. Section II provides an overview of quantum computing and QNNs. Section III discusses the general process for training, testing, evaluating, and tuning a QNN. Section IV provides details about our QNN framework for wireless network optimization. Section V explains experiment setup, details about how we trained and tuned our specific QNN, and provides advice about key tuning parameters. Section VI contains a detailed evaluation and discussion about our results. Detailed discussion of parameters such as: number of qubits, layer counts, and training batch size will be tuned and evaluated. In this section, we will establish that a QNN can arrive at a result comparable to a Traditional Neural Network (TNN). A TNN will also be used as a baseline to validate our QNN results and demonstrate the potential of quantum computing for wireless network optimization. Our paper concludes with Section VII, where we discuss potential for future applications of QNNs in the wireless communications space.

We refer readers interested in traditional or state-of-the-art artificial intelligence algorithms for optimizing wireless communications to recent examples in the literature: [5], [6], [7], [8], [9], [10], [11], [12]. A small neural network will be introduced Section V-C to use as an experimental baseline for our QNN.

II. OVERVIEW AND MOTIVATION FOR USE OF QUANTUM COMPUTING AND QUANTUM NEURAL NETWORKS

Quantum computing [2] and QNNs [1], [38], [39], [40] have emerged as two promising paradigms to solve complex computational problems that are intractable for classical methods. Quantum computers leverage the principles of quantum mechanics, such as superposition and entanglement, to process vast amounts of data simultaneously, offering a potential speedup in areas like optimization, cryptography,

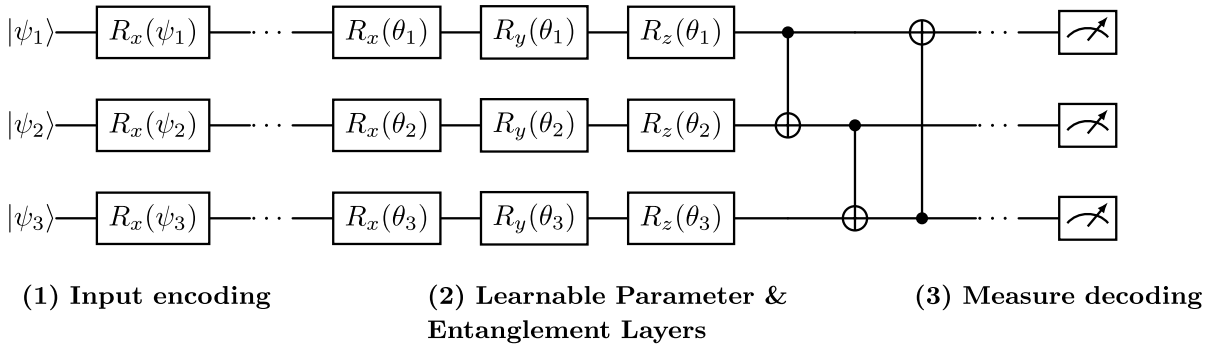

FIGURE 2. 3-qubit Quantum circuit for QNN Regression. A single layer consists of a series of Rx, Ry, Rz gates, fully entangled. Input encoding is performed by the first set of R_x gates (1). The following gates are where parameters are trained (2). The measure decoding step (3) allow extraction of data and results from quantum network. Details in Sections II-A & II-B.

and machine learning [35]. As transistor sizes in traditional computers grow smaller, "Quantum-effects" [42] begin to degrade transistor reliability and performance [43]. Quantum computing is a mode of computing that leverages the properties of quantum mechanics.

One key example of a quantum property is superposition, [44], [45] where quantum bits (qubits) exist in multiple states simultaneously, unlike classical bits that can only be in one of two states (0 or 1). Since qubits can represent both 0 and 1 at the same time, quantum computers can process vast amounts of information in parallel, exponentially increasing computational power for certain tasks. By utilizing this property of quantum mechanics, quantum computers can explore multiple computational states simultaneously, thereby potentially solving certain types of problems exponentially faster than classical computers. One relevant motivation for using quantum computing is its ability to handle high-dimensional data and solve complex optimization problems that would be infeasible for classical approaches due to resource constraints. In areas such as drug discovery, materials science, and large-scale simulations, quantum computers can provide an advantage by simulating molecular interactions and physical systems with unparalleled accuracy [2], [24], [25].

In wireless communications, quantum computing has the potential to revolutionize areas where data and interactions between systems becomes increasingly complex. There are already many examples in the open literature of researchers investigating areas such as Transport Network Design [4], Channel Estimation, Routing, Indoor Localization, and Data Analysis [14]. Quantum computing can also optimize massive Multiple Input, Multiple Output (MIMO) systems by efficiently solving high-dimensional matrix operations and complex beamforming problems, which are critical for enhancing the performance of 5G and beyond networks. Additionally, quantum computing can be applied to solve difficult combinatorial optimization problems in network routing and spectrum allocation, ensuring optimal use of resources in increasingly congested wireless environments [15], [16], [17], [33], [34].

The justification for using quantum computers and QNNs lies not only in their ability to solve previously intractable problems, but also in their potential to revolutionize industries by providing computational power beyond the reach of classical systems. As classical hardware improvements slow down and data become more complex, quantum computing stands out as a revolutionary tool for advancing research and development in a variety of fields, including machine learning, cryptography, and material science. As technology matures, quantum computing could transform industries by providing unparalleled speed and efficiency in solving some of the most computationally expensive problems known today [2].

Quantum computers are constrained by the number of qubits available and their susceptibility to noise. As the size and complexity of quantum circuits grow, the noise introduced by qubit interactions becomes more significant, potentially degrading the performance of quantum algorithms. For convenience, here is a list of the strengths and weaknesses of quantum computing:

STRENGTHS

- Exponential speed-up for specific algorithms (factoring, search, optimization).
- Efficient simulation of quantum systems.
- Potential improvements in machine learning.
- Quantum cryptography for secure communications.

WEAKNESSES

- High error rates due to noise and decoherence.
- Complex, costly, and sensitive hardware.
- Few demonstrated practical use cases.
- Difficulty scaling qubit count.

The exploration of QML in wireless communication applications has increased progressively in the past five

years. However, there is a need for detailed implementation information to be presented in the open literature. Furthermore, several of these published solutions are focused on a specific domain within wireless communications. For a comprehensive overview on the topic of quantum computing in the field of wireless communications, we refer the interested reader to [13], specifically the tables, which provide a detailed list of the many optimization problems in wireless communications which are being explored with quantum computing. For more examples and details about deployment of QML in wireless networks we refer readers to [14], [15], [16], [17], [18], [19], [20], [21], and [22]. A detailed theoretical explanation and implementation of QNNs from the perspective of wireless communications is presented in [18].

A. QUANTUM NEURAL NETWORKS

QNNs [1], [38], [39], [40], [41] are a popular quantum artificial intelligence technique, making them an ideal candidate for illustrating implementation of our architecture and methodology. The basis of a QNN is the concept of Variational Quantum Circuits (VQCs) [46], [47]. These are a type of quantum circuit designed to solve optimization problems by leveraging a series of parameterized quantum gates applied to qubits [48]. Gate parameters are tuned during training using classical optimization methods. This allows the quantum circuit to function similarly to a TNN and make predictions based on training data.

The theoretical advantages of QNNs include the potential for exponential speedup in processing complex tasks, the ability to operate in higher dimensional feature spaces through qubits, and more efficient handling of quantum data. Additionally, QNNs may offer resource-efficient solutions to certain optimization problems that are computationally intensive for TNNs. These benefits make QNNs a promising alternative for solving problems that classical methods struggle with. As mentioned briefly in Section II, these are not realized improvements. Quantum computers, and by extension QNNs, remain an experimental technology with promising theoretical advantages, but they are still limited by current hardware challenges, error rates, and scalability issues that need to be overcome before practical applications can be fully realized. It is well established in the open literature that QNNs can theoretically, or in a simulation, achieve equal or superior performance compared to a TNN [57], [58], [59], [60]. This allows us to confidently use a TNN as a baseline when implementing our own QNN.

In the field of wireless communications, QNNs have notably been discussed for 6G communications [15], [16], [17] and Resource Allocation [18]. As discussed in Section II, any application in wireless communications with a high-dimensionality dataset (*e.g.,* resource allocation, beamforming, 6G, route planning) could be significantly enhanced by the computational power of QNNs. Some exploration exists for reinforcement learning [46], [49] as well as generative networks [41]. Outside the field of wireless communications, QNN architectures exist for a variety of applications in open literature [30], [47], [50], [51].

B. EXAMPLE QNN ARCHITECTURE

The core architecture of a QNN consists of layers of quantum gates. Each layer employs quantum gates such as R_x, R_y, R_z, and controlled operations like CNOT. This QNN architecture is shown in Figure 2, and is a well-established architecture for QNNs [16], [52], [53]. The first layer of gates (1) in Figure 2 consists entirely of R_x gates for input encoding. This layer encodes the traditional dataset of real numbers into qubit state according to the circuit's qubit count. The next layer (2) in Figure 2 processes and learns from the input data. Each set of R_x, R_y, R_z, and CNOT gates in (2) is considered a single layer. The network is trained to adjust the parameters of these quantum gates to minimize the difference between predicted and actual output values. Like TNNs, layers can be stacked [54]. Simply increasing layers is not a viable solution, as QNNs are highly sensitive to arbitrarily large layer counts. They are prone to a well-known issue called the Barren Plateau Phenomenon [55], [56], in which an excessive number of qubits or layers leads to vanishing gradients. This means that the cost function's gradient becomes exponentially small across most of the parameter space, causing the training process to slow down significantly and making it nearly impossible for the optimization algorithm to find a good solution.

The final step is (3) measure decoding, where the encoding process is essentially reversed, and qubits values are measured and decoded. After this point, the outputs are in the realm of traditional computing. The next step could be to apply a technique such as regression, classification, or even clustering, depending on the nature of the data and the specific goals of the network.

C. TOOLS FOR QUANTUM NEURAL NETWORK IMPLEMENTATION AND EMULATION

As discussed in Section II-A, available literature has shown that QNNs can perform as well as or even outperform TNNs. Given the unreliability, noise, and cost associated with quantum hardware, researchers may opt to test their QNNs in an emulator to ensure the soundness of their network and implementation. Emulation on traditional hardware is computationally resource-intensive, but not as cost-inhibitive as deploying a QNN to real quantum hardware. An emulation-first approach allows researchers to establish as a baseline that their techniques work before using time on a real quantum computer. Once a network is proven to be effective in an emulator, one might then progress to a simulator, or quantum hardware to debug and evaluate real-world performance. A major downside of this approach is that computational performance will be worst. Getting responses from an emulated QNN will inherently be slower than getting

TABLE 1. Overview of quantum computing libraries.

Library	Description	Key Features	Notes and issues	Reference
Qiskit	An open-source SDK by IBM for programming quantum computers, simulators, and emulators in Python.	Python-based, first-class IBM quantum hardware support. Many tutorials, very popular.	Moderate pytorch and tensorflow library support. Limited hardware options compared to other libraries and tools.	[96]
Cirq+TensorFlow Quantum (TFQ)	Cirq is a library for running quantum simulations, with TFQ integrating QML using TensorFlow primitives.	Tight TensorFlow integration, quantum circuit creation, built-in Google hardware simulators.	Primarily designed for Google's quantum processors. TFQ software on GitHub is infrequently updated, with gaps of over a year between releases	[89]
PennyLane	Python library for quantum ML, integrating classical ML with quantum computing.	Cross-platform, hybrid models, integrates reasonably well with frameworks like PyTorch and TensorFlow.	Many outdated tutorials. Quality of pytorch or tensorflow integration varies depending on application. Focused heavily on hybrid quantum-classical algorithms.	[97]
TorchQuantum	PyTorch-compatible library for QNNs with GPU-accelerated emulations.	High quality Pytorch integration. Large-scale emulations, classical optimization. Simplified API to facilitate easy transition for machine learning practitioners.	Active development, certain tasks are unstable. Limited resources and documentation compared to other libraries.	[88]

responses from a TNN. This is due to the overhead cost associated with emulation.

To address some of the challenges described earlier in Section II, and to simplify the process of quantum circuit creation, the TorchQuantum library is used within the framework of this paper [61], [62]. Table 1 shows a list of libraries that one might use for quantum machine learning. All of these libraries are viable options and selection will be dependent on an experimenter's requirements and preferences.

TorchQuantum was selected for this paper because it enables fast prototyping with PyTorch primitives in an emulator [88]. TorchQuantum abstracts away many of the complexities of quantum circuit design, making it more accessible to machine learning practitioners who are familiar with PyTorch, thus facilitating a more rapid development of QNNs. These features enable users to quickly and easily explore the potential speedups and advantages that quantum computing could bring to machine learning tasks, without having to make major changes to their datasets.

One of the primary strengths of TorchQuantum is that it also allows quantum networks to be run in emulation, leveraging classical hardware to mimic quantum operations. The library provides an emulation environment where researchers can validate their models and experiment with different quantum circuit architectures without the need for immediate access to quantum computers. This emulation capability is essential for developing and testing quantum algorithms. As mentioned earlier in this section, it reduces the cost of experimentation and allows users to explore the performance of their algorithm without having to accommodate for quantum noise and error [36].

TorchQuantum does not natively support quantum hardware, but it integrates with the Qiskit library [90] for streamlined deployment. Once a quantum network is developed and tested, it can be converted to and directly executed on Qiskit-compatible quantum processors [89]. This capability allows researchers to move seamlessly from experimentation in an emulated environment to real-world applications on quantum devices, thus shortening the development cycle for QML solutions [89], [90].

Other libraries and tools shown in Table 1 are viable options depending on the application. For practitioners more accustomed to the Tensorflow library, Tensorflow Quantum accompanied by Cirq may be a better alternative. Qiskit is another very popular library shown in Table 1, although it has limited hardware options outside of IBM's quantum hardware ecosystem. Given the advantages of fast prototyping, emulation, and rapid deployment, TorchQuantum is an appropriate solution with respect to exploring the potential of quantum machine learning for wireless networks. Users should consider their own needs when selecting the appropriate library for their research.

III. QNN TRAINING, TESTING, TUNING, PERFORMANCE

Given the TorchQuantum library, the data pipeline for training a QNN closely follows that of a classical neural network. Input and output features are preprocessed using normalization techniques such as MinMax scaling [92]. The dataset is split into training, testing, and validation sets, thus enabling the evaluation of a model's performance on unseen data. For a machine-learning practitioner new to training QNNs, parameters of particular interest in a QNN will be:

- Qubits - the number of qubits in the QNN
- Layers - the number of layers in the QNN
- Batch size - the size of each training batch shown to the QNN

Similarly to a TNN, standard loss functions such as mean squared error (MSE), mean absolute error (MAE), binary cross-entropy (BCE), and categorical cross-entropy (CCE)

can be applied. To learn more about Loss Functions, the authors recommend to the interested reader the following references [65], [66]. Additionally, a wide range of popular optimizers can be used, including Gradient Descent and its variants, as well as Adaptive Learning Rate methods [67], [68]. The choice of loss function or optimizer depends on the dataset, neural network layout, and specific objectives.

Tuning and evaluating the performance of the QNN is done by evaluating the network's accuracy against test data. For a detailed understanding of the tuning process, the QNN's performance should also be compared to a TNN or known algorithm:

1) Train QNN with a given dataset,
2) Generate predictions from trained QNN,
3) Evaluate QNN predictions against predictions generated by a known TNN or established algorithm,
4) Modify a parameter (qubits, layers, batch size) and train QNN again,
5) Repeat training and tuning in above steps until satisfied with results.

Per Step (3) in the tuning steps described above, we must define which evaluation techniques will be used when comparing a TNN and a QNN. Two common evaluation techniques are Euclidean distance and cosine similarities. These two metrics are often used together to analyze data and offer complimentary insights [69], [70], [71], [72]. Euclidean distance (Eq (1)) is an effective metric for comparing the outputs of two neural networks in regression tasks, as it directly quantifies the geometric distance between corresponding points in Euclidean space. This metric excels at measuring the magnitude of differences between output values, offering a clear and interpretable indication of the proximity or disparity between the outputs [73], [74]. A smaller Euclidean distance value indicates greater similarity.

In N-dimensional space, with a vector consisting of x_i elements representing QNN outputs, and a vector consisting of y_i elements representing TNN outputs, the distance $d(\mathbf{x}, \mathbf{y})$ can be calculated with Eq (1). For any given set of outputs, the difference in magnitude between a prediction from a QNN and a TNN can be compared.

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{1}$$

Cosine similarity (Eq (2)) captures directional alignment of the same vectors, x_i and y_i in N-dimensional space. As we can see in Eq (2), cosine similarity is a normalized value. Results are fixed between -1 and 1. While Euclidean distance measures absolute magnitude in distance, cosine similarity is better at detecting semantic differences between vectors [69], [75], [76]. Values of 1 indicate perfect alignment, while values of -1 indicate complete opposition of vectors.

$$S_c(\mathbf{x}, \mathbf{y}) = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \sqrt{\sum_{i=1}^{n} y_i^2}} \tag{2}$$

Together, Euclidean distance and cosine similarity are useful as an evaluation metric because for any tuning change, we will be able to measure change in magnitude as well as directionality. Depending on the dataset and goals of the QNN, one technique may be more important to optimize than the other.

By measuring and tracking both Euclidean distance and cosine similarity throughout the tuning process, the influence of one metric over the other may be discovered. In such cases, the appropriate metric should be optimized accordingly. These two metrics are able to provide complimentary insights into data and both should be used in tandem throughout the tuning process to achieve optimal results.

IV. FRAMEWORK FOR OPTIMIZING QNN-OPTIMIZED WIRELESS NETWORKS

In this section, we present our proof-of-concept QNN framework that can be employed to perform wireless network optimization. This implementation serves as an initial step towards exploring the potential advantages of quantum machine learning in network optimization. By integrating the QNN within a wireless communication application, we demonstrate how quantum computing can be leveraged to enhance various decision-making processes. We also discuss the specific architecture of the QNN, and the methods used for training and evaluation. Subsequently, we explain how its performance compares to that of an equivalent TNN in Section VI.

The goal of this framework is to provide both practitioners and researchers alike within the field of wireless communications with a reference implementation of a QNN that can be adapted and employed in their own applications. The impact of various parameters on the QNN and its performance will also be explored as we demonstrate the feasibility and effectiveness of QNNs in real-world wireless network applications.

A. SYSTEM BLOCK DIAGRAM

Figure 1 illustrates our high-level architecture of the QNN-based wireless network. The design is generalized to support a wide range of wireless networks with varying degrees of complexity. Previous studies have introduced architectures that integrate quantum computing and quantum machine learning [3], [4], [15], [16], [17], [19], while other studies have focused on the theoretical aspects of quantum machine learning when exploring various QNN architectures and implementations [18]. By contrast, our proposed framework emphasizes the practical application of this architecture, enabling practitioners across the wireless community to develop, deploy, and optimize their own QNNs within diverse network environments. A brief description of each step in Figure 1 is given as follows:

1) Clients attempt to connect to a wireless network with limited or constrained resources (channels or spectrum, for example).

2) Network and clients apply a ruleset to allocate resources (*e.g.,* wireless channels) similar to protocols used in various wireless standards, such as WiFi [77], Bluetooth [78], Zigbee [79], LoRaWAN [80], or DECT [81].
3) Wireless network saves relevant connectivity data in a database.
4) Connectivity data is used to train the QNN.
5) Training is complete and the model is deployed. The size of the model will help dictate the nature of the wireless network and ruleset.
   - If the model is small enough, the ruleset can be deployed close to the wireless network, so ruleset decisions can be made quickly and frequently.
   - If the model is larger, the ruleset can be deployed in a way that introduces higher latencies with less frequent updates to rulesets.
6) The resulting QNN has learned the intricacies of the network, and can suggest new rulesets to better optimize utilization of the resource-constrained wireless network based on the requirements of the operator.
   - Note: Depending on the wireless network and protocol design, the ruleset may be partially or fully shared with clients through the access point server.

Given this high-level architecture, along with evaluation techniques and operational flow of our QNN-based wireless network optimization system, we now delve into implementation details when applying this framework to a specific wireless networking use case, namely of a generic CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) algorithm. Without loss of generality, the approach presented can be employed in other wireless use cases.

V. EXPERIMENTATION AND RESULTS: CSMA/CA USE-CASE

This section provides a comprehensive overview of how the proposed architecture and methodology can be implemented for a specific wireless application. To validate the efficacy of our QNN-based optimization strategy, we conducted a series of experiments using a simulated wireless network. The network model, based on a generalized version of a priority-based CSMA/CA protocol [82], allowed us to generate a diverse dataset capturing the connectivity behavior of clients under various rulesets. The network was designed to allocate channels to clients who were attempting to connect. Connection acceptance/rejection for open channels was decided by a ruleset. When a client failed to connect, it was granted a retry time based on its priority (high, medium, low). Client retry time was dictated by the ruleset, where higher priority clients would be allowed to attempt connections more frequently, while low priority clients would be allowed to attempt connections less frequently.

By training both a TNN and a QNN on the same dataset, we aimed to compare their performance in predicting optimal retry times for different client priority levels. The training goal for both models was to predict an optimal retry that would ensure the desired connection probability for different classes of clients. Quality of predictions were compared by using the evaluation methods initially discussed in Section III.

The experimental process involved several key steps as described in Section III:

1) Train QNN with a given dataset.
2) Generate predictions from trained QNN.
3) Evaluate (using Euclidean distance and cosine similarity) QNN predictions against predictions generated by TNN.
4) Modify a parameter (qubits, layers, batch size) and train QNN again.
5) Repeat training and tuning in above steps until satisfied with results.

Before the experiment can begin, the following items must be discussed in greater detail:

- Priority-based CSMA/CA simulation (see Section V-A).
- Data Generation (see Section V-B).
- Neural Network used as a baseline must be trained and tested (see Section V-C).

Having laid the groundwork for our QNN-based architecture, we now turn our attention to the practical application of this approach, specifically demonstrating its implementation in a priority-based CSMA/CA wireless network simulation. Examples are available in our code repository [83].

A. EXPERIMENT SETUP AND SIMULATED NETWORK

Our priority-based CSMA/CA network was executed as a simulation on a conventional computer workstation. A typical workstation for our tests was conducted using an Intel Xeon CPU with 2.2GHz clock speed, and 50GB of system RAM. For tests with very high qubit counts (shown in Table 4), a server with an AMD EPYC CPU at 3.7GHz clock speed and 1.6TB of RAM was used. Each network had 20 clients attempting to connect to a server which had 5 channels. There were 7 high priority clients, 7 medium priority clients, and 6 low priority clients. Without a model, the ruleset was configured manually. Additional information regarding the specific ruleset used for generating data is presented in Section V-B.

Connection Probability P_{conn} of a given client is described by Eq (3), where N_{fail} is the number of rejections that client received, and k is the number of connection attempts made. This metric can be used to calculate the exact performance of each client type given a selected ruleset:

$$P_{\text{conn}} = 1 - \frac{N_{\text{fail}}}{k} \tag{3}$$

Between the ruleset and our connection probability metric, we can analyze the results of each simulator run. Figures 3 and 4 illustrate the results of a single 30 second simulation run with 20 clients and 5 channels, where the clients are split roughly evenly, with 7 high priority clients, 7 medium priority clients, and 6 low priority clients. The ruleset in this

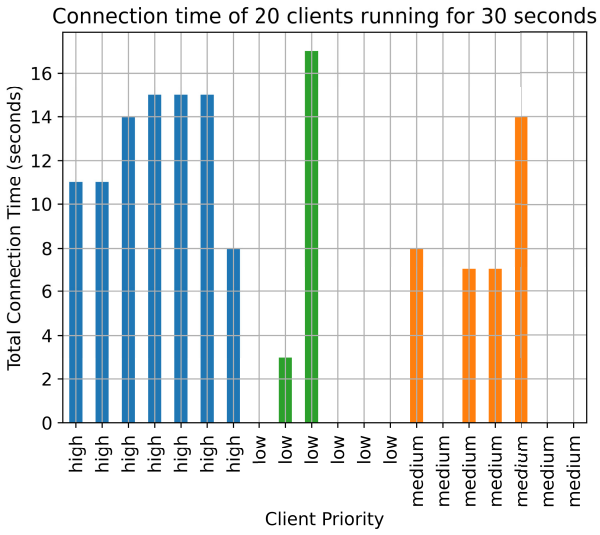

FIGURE 3. Example of total connection time of each client in simulation. All high priority clients were able to connect. Most medium priority clients were able to connect, and only two low priority clients were able to connect. The system is working as expected.

specific example permits higher priority clients to retry after a failed connection attempt more frequently than lower priority clients.

In Figure 3, we can see that the high priority clients all managed to connect for at least several seconds during the 30-second window. Only a handful of the medium priority clients were able to form a connection, and only two low priority clients managed to connect. This is desired behavior given the nature of our priority-based ruleset. The high priority clients are more likely to spend time connected to the server.

Figure 4 shows more detailed statistics about the simulation. The results are as expected - the high priority clients spent the most time connected, while the medium and low priority clients performed progressively worst. The high priority clients had the best overall performance - they were most likely to achieve a connection, were able to make more frequent connection attempts, spent the most time connected, and spent the most average time connected. None of the high priority clients failed to make a connection. Based on the retry rules shown in Table 2, we can see the probability of a high priority client forming a connection was 34.61%. For medium priority clients, this number was 20.41%. Low priority clients had a 14.36% chance of forming a connection. This table validates the expected behavior: higher priority clients receive preferential treatment and are more likely to successfully connect, while lower priority clients receive a lower quality of service. These metrics prove that our simulation is behaving in a predictable and consistent manner.

The following steps directly align with the generalized process outlined in Section IV-A and Figure 1:

1) 20 clients (7 high, 7 medium, 6 low priority) attempted to access 5 channels available at an access point server.
2) If channels are available, the client can connect and transmit data. If no channels are available, the client is rejected and can retry after a timeout defined in the ruleset. The ruleset is fixed and unchanging for a simulation run. An example ruleset:
   - High priority client can retry in 2 seconds
   - Medium priority client can retry in 5 seconds
   - Low priority client can retry in 7 second
3) Data from each simulation run is stored.
4) Data is used to train a machine learning model for regression. In our specific experiment, we trained both a TNN and a QNN to evaluate performance of the QNN.
5) Models are trained to select appropriate retry times, based on desired connection probabilities.
   - "Deployment" in the case of our model was simply that it was available for us to query in our codebase.
6) The model is queried to provide rulesets that will allow the network to achieve the desired performance.

With the experiment set up and simulation parameters defined, the next step involved generating the necessary dataset by repeatedly running the simulation with various rulesets and storing the resulting connectivity data for further analysis.

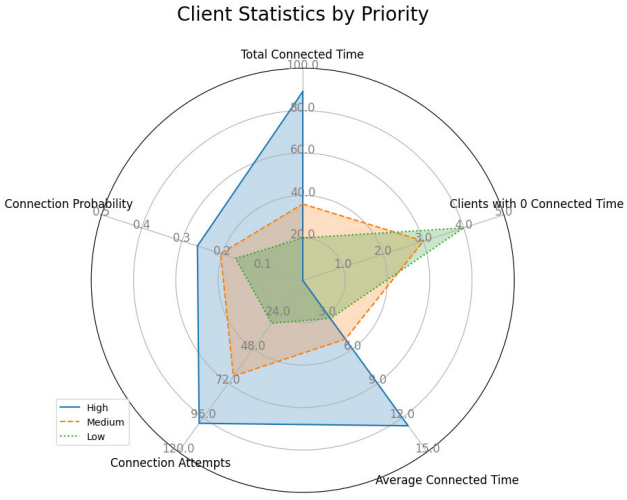

FIGURE 4. Client statistics by priority. High priority clients have the best performance for metrics such as Connection Probability, Connection Attempts, Average Connected Time, Clients with 0 connected time, and Total Connected Time. Medium priority clients performed slightly worst. Low priority clients had the worst performance.

B. GENERATING DATASET

Our dataset was generated by executing our network simulation 1500 times with human-selected rulesets. The large number of runs was to ensure we had ample data to train our neural networks. For each simulation, we assign a connectivity rule set that dictates which types of clients could connect and under which conditions. Clients that

TABLE 2. Example retry time and average connection probabilities for a single simulator run.

Worker Priority	X, Retry Times (sec)	Y, P_{conn}
High	1.7002	0.3461
Medium	3.7157	0.2041
Low	5.6736	0.1436

failed to connect were given a retry time, before which they could not make another connection attempt. Each simulation generated data describing the assigned ruleset for that simulation and the connection probability of each client for the duration of the simulation. By executing 1500 simulations and saving the results, we generated a dataset which described the connectivity behavior of the clients across a variety of conditions. The data relevant to our models:

- Client Priority (from ruleset)
- Retry times after connection failures (from ruleset)
- Connection Probability of each client (from simulation data)

Table 2 shows an example data set from a single simulation run. Although the simulator is not fully deterministic, it consistently produces results that are similar and statistically converge to the same average outcome over multiple runs. Thus, we generated a large dataset where the features **X** are the retry times for each simulation run, and the labels **Y** are the average connection probabilities for each class of client corresponding to that run.

C. TRADITIONAL NEURAL NETWORK AS A BASELINE

To establish a performance baseline in the traditional computing space before exploring the potential of a QNNs, we trained a TNN to perform regression across three parameters in our dataset (high, medium, low priority retry times). This benchmark allowed us to compare the performance of our QNN against a well-established and widely-used machine learning approach, using our evaluation techniques from Section III.

To perform regression across three parameters, a simple feed-forward neural network with 4 fully-connected hidden layers was used. A more complex or deep neural network was not used due to the simple structure and relatively low volume of the data. Studies show that a Deep Neural Network (DNN) with many layers is not required to achieve good performance for small, simple, low-dimensionality datasets [63], [64]. The input layer and output layers each had 3 input features: one for each priority level. For forward propagation, we used the ReLU activation function for the hidden layers and a linear activation function for the output layer.

We used Mean Squared Error loss function and an Adam optimizer with a learning rate of .001 to train the network. The training process was run for 250 epochs. A dataset consisting of 1,500 simulations, each with a unqiue ruleset were used for training/testing/evaluation. Training took a few minutes on modern hardware. We achieved a training loss of 0.047 and a test loss of 0.049.

TABLE 3. Target connection probabilities, actual connection probabilities, and retry times selected by the neural network.

Desired Connection Probability (High, Medium, Low)	Observed Connection Probability (Avg of 500 runs) (High, Medium, Low)	Predicted Retry Time (High, Medium, Low)
0.3, 0.2, 0.05	0.35, 0.20, 0.14	1.06, 3.8, 6.1
0.4, 0.2, 0.1	0.44, 0.18, 0.11	0.96, 3.9, 6.28
0.2, 0.2, 0.2	0.18, 0.27, 0.23	3.65, 3.76, 3.3
0.6, 0.3, 0.1	0.45, 0.22, 0.18	6.16, 8.21, 8.33

Once the neural network was trained, we used it to suggest optimal retry times for each priority level, given a desired connection probability. We tested the quality of these predictions by running the simulation with the suggested back-off times and comparing the actual connection probability to the desired connection probability.

Based on results presented in Table 3, the observed connection probabilities are close to the expected values, demonstrating a high level of consistency between the model's predictions and the actual outcomes. Thus, we can rely on this TNN as a benchmark for our QNN.

Sections V-D2 and V-D3 use the TNN's predictions as a baseline to discuss the effects of each parameter on QNN accuracy. Euclidean distance and cosine similarity metrics presented in Section III will be applied to compare the QNN's predictions against those of the TNN. Section V-D4 will providing a summary of optimal QNN Architecture Parameters selected by this evaluation and tuning process.

D. TUNING QNN PERFORMANCE

The QNN architecture used for this experiment is exactly as shown in Section II, Figure 2. It was trained with the same dataset used to train TNN discussed in Sections V-B and V-C. The same loss function (Mean Square Error) and Adam optimizer as the TNN were also used. As described in the beginning of Section V, the network was trained, then evaluated against the TNN, and then trained again with a different training parameters each time. We developed a simple parameter sweeping script to train the (QNN) across various combinations of qubit counts, layer counts, and batch sizes. The predictions generated and simulation performance of each combination was written out to a file to be analyzed at the end of a sweep. The optimal configuration was selected based on the comparison of Euclidean distance and cosine similarity against the performance of the TNN.

1) IMPACT OF QUBIT COUNT ON HARDWARE UTILIZATION AND TRAINING TIME

Training the QNN proved to be significantly more resource intensive than the TNN. Notably, as the number of qubits increased, RAM usage grew exponentially, as presented

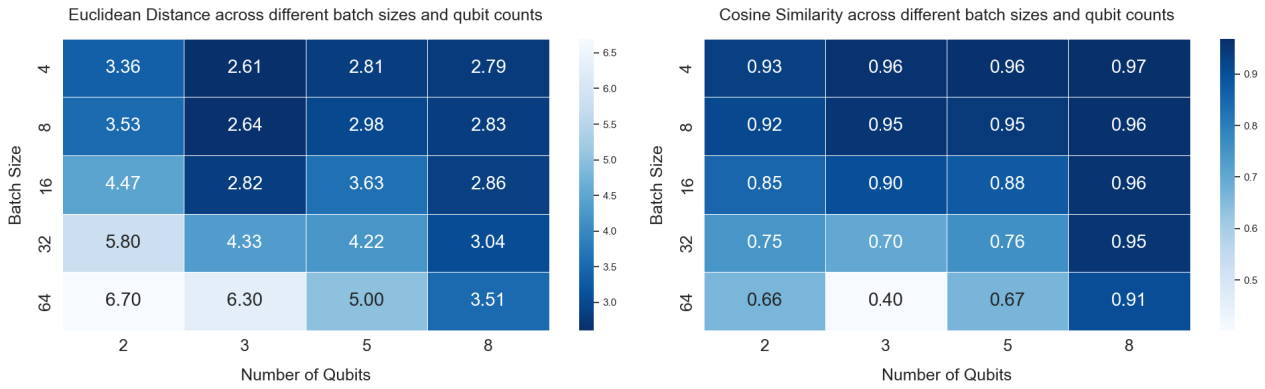

FIGURE 5. Euclidean distance and cosine similarity between QNN and TNN predictions, for varying qubit counts and batch sizes. Euclidean distance and cosine similarity generally agree on which configurations are best. Smaller batch sizes tend to improve performance. Increasing qubit count can have a positive impact on cosine similarity, but the return is diminishing.

in Table 4. This finding is consistent with other works presented in the open literature [50], [84], which highlights the steep computational costs of scaling quantum systems. For instance, the jump from 10 qubits (800MB RAM) to 12 qubits (81GB RAM) represents a 100-fold increase in memory requirements, illustrating the steep resource curve. Additionally, training time showed a marked slowdown with the increase in both qubits and network layer, further underscoring the substantial computational demands associated with scaling QNNs.

It should be noted that Table 4 presents a practical limitation on the size and complexity of QNNs that can be emulated on classical hardware. It also demonstrates how rapidly hardware and time requirements escalate with qubit count. To mitigate scaling challenges, it is preferable to minimize the number of qubits and leverage the benefits of small batch sizes coupled with large layer counts.

TABLE 4. RAM usage and training time for various qubit configurations.

Qubits	Layers	Peak RAM Usage	Training Time
2	20	49MB	21s
3	20	103MB	31s
5	20	254MB	54s
8	20	586MB	1m 50s
10	20	800MB	3m 11s
12	20	81GB	4m 39s
15	20	233GB	6h 44m
20	20	580GB	22h 46m
21	20	1.21TB	24h 9m

Note: Table 4 shows networks trained only with 20 layers, because an increase in layers by a single order of magnitude at high qubit counts exponentially increased RAM consumption and significantly increased training time. Therefore, it was not used for the timing and resource allocation experiments. A layer count of 20 sufficiently illustrates the drastic increase in resource consumption.

2) IMPACT OF QUBIT COUNT AND BATCH SIZES ON QNN PERFORMANCE

Figure 7 shows the relationship between qubit count, batch size, and network performance. Batch size controls the number of training samples used in each iteration of the neural network's training process, and is an important tuning parameter in TNNs [91]. The Euclidean distance and cosine similiarities between QNN and TNN predictions were plotted as a heatmap. The Euclidean heatmap shows that smaller batch sizes and fewer qubits produce the best results (3 qubits, batch size of 4). 8 qubit networks also demonstrated acceptable performance, however, smaller qubit networks consume less hardware resources.

The cosine similarity heatmap shows a similar trend, where smaller batch sizes produce the best results. There are minor discrepancies between the two heatmaps. cosine similarity shows minor improvements by using 8 qubits instead of 3 for batch sizes of 16, 8 and 4. Since there is a 1% difference in cosine similarity for the best-case scenarios (3 qubits versus 8 qubits for batch sizes of 4) we elect to use the configuration parameters where our two metrics agree, and hardware utilization is lower.

The observation that increasing the number of qubits does not necessarily lead to improved performance is consistent with findings report in literature [47], [85]. In scenarios where data complexity is high, fewer qubits may be insufficient. One proposed solution is a Distributed QNN, where the data and corresponding QNN are partitioned into multiple smaller datasets and QNNs [86], [87].

Figure 5 establishes that networks trained with smaller batch sizes tend to achieve better performance. This was especially true for networks with smaller qubit counts. The primary drawback of smaller batch sizes is the increase in training time. As shown in Figure 6, the improved performance associated with smaller batch sizes comes at

a significant cost. For example, training a 3-qubit network with a batch size of 4 takes approximately 15 minutes. This is manageable for a small, three-dimensional dataset, larger datasets or more complex networks may render very small batch sizes impractical.

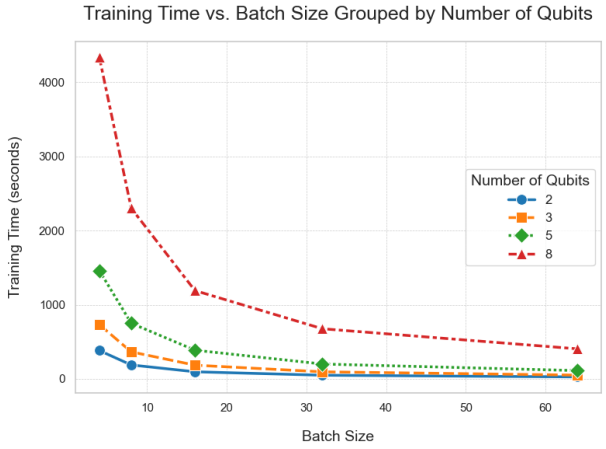

FIGURE 6. Training time versus batch size, grouped by qubit count. Convergence is observed, but the fastest training time came from smaller qubit counts.

3) IMPACT OF LAYERS ON QNN PERFORMANCE

As shown in Figure 2, one layer of the network consists of a block of R_x, R_y, R_z, and CNOT gates [61]. Layer counts were increased across training runs to evaluate their impact on performance. The results are shown in Figure 7. They indicate that increasing the layer count initially leads to significant performance improvements, but these gains diminish rapidly as the number of layers grows. For our specific dataset, we observed this plateau occur between 100 and 200 layers. Euclidean distances drop sharply until around 100 layers. After this, we observe a diminished or even negative improvement.

These trends continue with the cosine similarity metric. With the exception of 2 qubit networks, which categorically perform worst beyond layer counts of 20, cosine similarity converges until roughly 100 layers. For 3 qubit networks, we see a 2% improvement between 100 and 200 layer networks. Cosine similarity of 5 qubit networks diverges by 1%, and for 8 qubit networks it remains the same.

Based on Figures 5 and 7, the overall best performance would be achieved by using a 3 qubit, 200 layer network. It achieved the best performance in terms of Euclidean distance (2.61), and a second-best cosine similarity (0.96). While 5 qubit and 8 qubit networks were trained and achieved similar performance characteristics, a 3 qubit network would be preferred due to its faster and less resources-intensive training times, as discussed in Section V-D1.

Larger layer counts (400, 1000, 2000) were briefly explored, but not examined in detail, as they significantly increased training time and resource usage without yielding notable performance improvements. On actual Quantum Hardware, very large layer counts are not desirable, as they have been found to exhibit degraded performance [60]. Specifically, very large layer counts result in the Barren Plateau Phenomenon mentioned earlier in Section II-A [55], [56].

4) OPTIMAL QNN ARCHITECTURE PARAMETERS

We found the following parameters to give us overall best performance:

- Qubits: 3
- Layers: 200
- Batch size: 8

This setup achieved a training loss of 0.062 and a test loss of 0.065. Based on the evaluation methods of Euclidean distance and cosine similarity this configuration achieves accurate results while maintaining a reasonable computational cost and training duration.

Three qubits were selected as they provided a good balance between computational resources and did not cause any significant loss in performance as compared to five and eight qubit networks. A batch size of eight was optimal because it allowed for high accuracy without significantly slowing down training time. The layer count was set to 200 as it was found to provide the best overall performance for the three qubit configuration.

Table 5 summarizes the final configurations for the TNN and the QNN. The QNN benefited from having smaller batch sizes, more layers, and fewer epochs. The TNN was able to achieve good performance with a much higher batch size. Layers between the neural network types are not equivalent and have been differentiated as "Traditional Layers" and "Quantum Gate layers." As shown in the quantum circuit from Figure 2, each "layer" in a QNN consists of quantum gates, and is not the same as a conventional layer of neurons in a TNN. Since QNNs use the same training techniques as TNNs, entries like batch size, learning rate, loss function, and optimizer are directly comparable. Fields such as qubits, conventional layers, and quantum gate layers, are not comparable since both types of neural networks do not use them, and are marked as N/A for such cases in Table 5.

TABLE 5. Comparison of TNN and QNN configurations.

Metric	TNN	QNN
Batch Size	32	8
Learning Rate	.001	.001
Loss Function	MSE	MSE
Optimizer	Adam (0.001)	Adam (0.001)
Qubits	N/A	3
Conventional Layers	4	N/A
Quantum Gate Layers	N/A	200

By carefully tuning these parameters, a high prediction accuracy was achieved. In Section VI, we will evaluate the

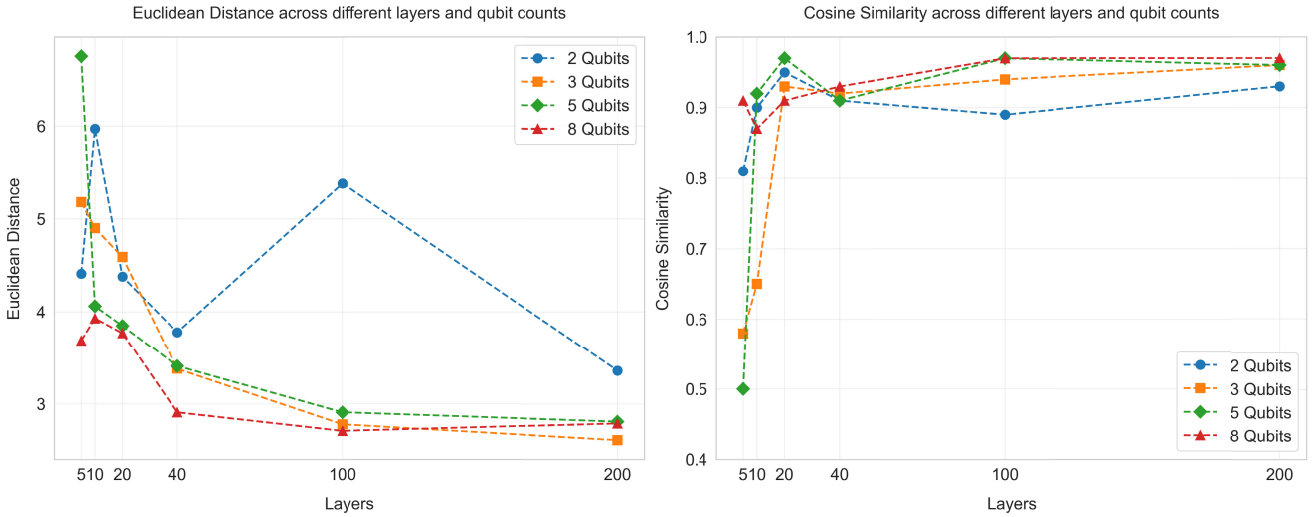

FIGURE 7. Euclidean distance and cosine similarity for varying qubit and layer counts. Significant fluctuations in Euclidean distance are observed for 2-qubit networks. Networks with 3, 5, and 8 qubits show convergence as the number of layers increases, with higher-qubit networks converging more rapidly.

TABLE 6. TNN and QNN retry time predictions for desired connection probabilities of 0.3, 0.2, and 0.05 for each type of client.

Priority	Connection Probability	TNN Prediction	QNN Prediction
High	0.30	1.06	1.20
Medium	0.20	3.80	2.60
Low	0.05	6.10	5.00

predictions made by the TNN and QNN, and compare their performance in our simulator.

VI. EVALUATION AND COMPARISON

As shown in Section V-D, the QNN could be made to converge to comparable performance to the TNN after parameter tuning. To rigorously assess its consistency, one thousand simulation runs were conducted for each type of neural network. These were the steps for evaluation:

1) Input a set of probabilities into the TNN.
2) Input the same set of probabilities into the QNN.
3) Run one thousand simulation runs for each set of predictions times generated in Steps 1 and 2.
4) Compare the connection probabilities of clients in each simulator run.

For the remainder of the results section we will discuss the set of probabilities and retry times shown in Table 6. Any reasonable values can be used. These values were selected merely as examples. Inspection of each individual prediction does not yield any insights. The predictions for high priority clients are quite similar between the TNN and QNN (1.06 versus 1.20, respectively). The predictions for medium priority workers are more than 1 second off (3.80 versus 2.60). The same is true of the low priority workers (6.10 versus 5.00). Our evaluation techniques from Section III, however, indicate that these results are quite similar. The Euclidean distance between the TNN and QNN predictions is 1.63, and the cosine similarity is 0.995. Therefore, we expect the QNN's performance in our simulator to be similar to the TNN's. We will investigate this empirically in a simulation in Section VI-A.

A. EMPIRICAL EVALUATION OF PREDICTION ACCURACY

As described in Section VI, once we have predictions for retry times from the TNN and QNN, these predictions are run in the simulator. Figure 8 allows us to visualize the alignment in performance between the TNN and the QNN, for each type of client. Five metrics are compared for each: Total connected time, clients with 0 connected time, connection probability, connection rejected count, and connection attempts. The alignment for all client types is almost exact. The greatest differences are in connection attempts and connection reject count. The QNN-managed ruleset made less frequent connection attempts, and was less frequently rejected. The total connected times are almost the same, and the connection probabilities varied slightly. In all cases, the QNN connection probabilities were slightly higher than the TNN. A higher connection probability is not the desired behavior in this case. The objective is to achieve the exact target probability requested for each client type. Both networks failed at keeping the low priority workers close to the 0.05 mark requested. However, they both achieved similarly poor performance in this area. The TNN achieved a slightly better connection probability for low priority clients at 0.14 than the QNN at 0.15. This indicates that while both networks were wrong, the QNN's performance is consistent with the TNN. Both networks roughly matched their target connection probabilities for

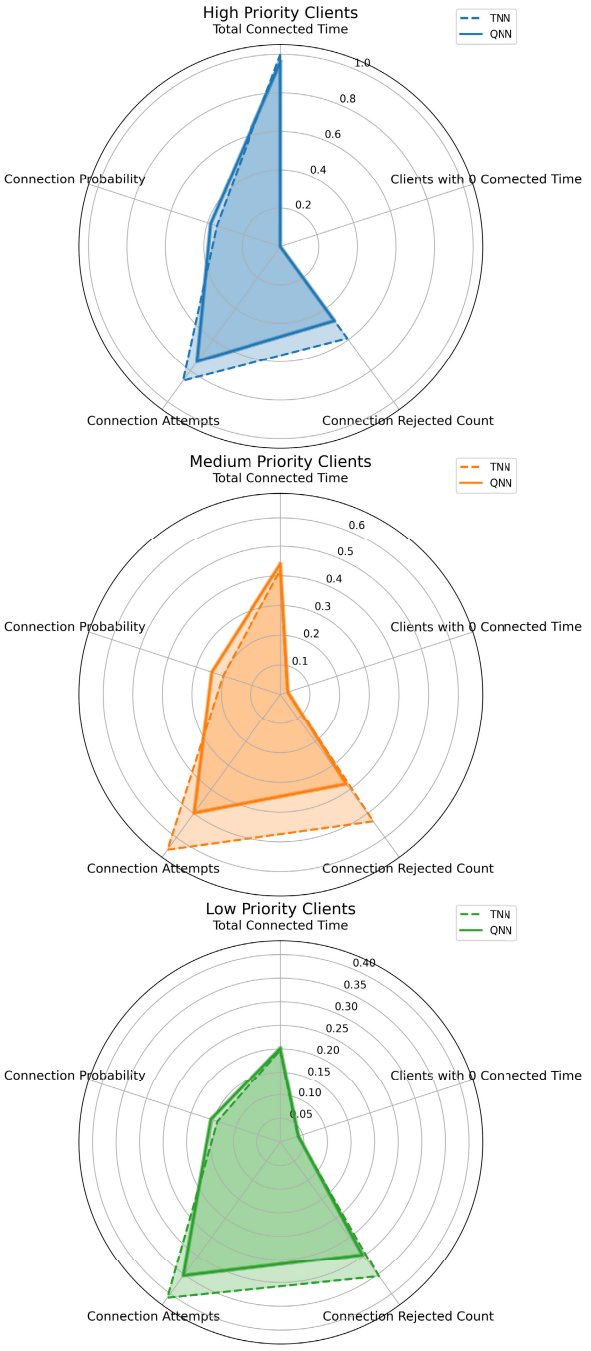

FIGURE 8. Results from experiments across 1000 simulation runs for the TNN and QNN. Performance pattern for each client shows similar responses between TNN and QNN. All values normalized for plot consistency.

high priority workers. The TNN's medium priority workers achieved the desired connection probability, while again the QNN's medium priority workers slightly overshot.

The graph in Figure 9 also allows us to inspect the spread of client connection probabilities for all 1000 simulation runs. This graph illustrates the variations and consistency across all runs. The spread is largely symmetrical, again indicating consistent performance between the TNN and QNN.

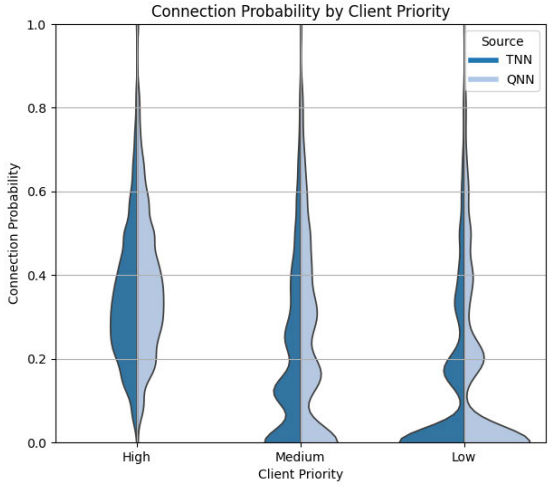

FIGURE 9. Connection probabilities across 1000 simulation runs for QNN and TNN. Shape of responses from TNN and QNN are similar, indicating similar performance across all runs for the two networks.

VII. CONCLUSION AND MOVING FORWARD

In this work, we introduce a framework that employs a generalizable system architecture designed to integrate a QNN into wireless networks. With this framework, and the authors' source code available at [83], we have demonstrated a successful integration of a QNN within a simulated wireless network, highlighting its potential for future applications in network optimization. With careful tuning across many parameters, the QNN presented in this paper was able to achieve, in emulation, similar results to a TNN. We found that qubit counts are severely limited during emulation, due to memory and time constraints during training.

QML holds as-yet unrealized potential to outperform traditional machine learning in tasks requiring the processing of data within highly complex and nonlinear systems. This framework provides the wireless communications community with a pathway to integrate quantum models into wireless networks and assess their viability in addressing pressing challenges. Through constructing a QNN and embedding it within a simulated network, we showcased the process of tuning and evaluating QML in an emulator, thereby exploring its feasibility and impact in real-world applications.

This paper lays the groundwork for a variety of wireless communications efforts that could benefit from QNNs, including optimizations in the physical, data link, and MAC layers. The insights presented here better equip practitioners to leverage QML in solving complex challenges in wireless communications. Key applications include: handling rapidly time-varying channels, optimizing large antenna and MIMO arrays, improving channel estimation, interference mitigation in dense networks, adaptive beamforming, access control optimization, or power amplifier distortion compensation.

As demonstrated in this paper, QML paves the way for enhanced performance, reliability, and efficiency of modern

and future wireless communication systems. We hope to see practitioners apply the proposed framework and tuning process in their own wireless networks.

REFERENCES

[1] N. Abdelgaber and C. Nikolopoulos, "Overview on quantum computing and its applications in artificial intelligence," in *Proc. IEEE 3rd Int. Conf. Artif. Intell. Knowl. Eng. (AIKE)*, Dec. 2020, pp. 198–199.

[2] McKinsey & Company. (2024). *The Rise of Quantum Computing*. [Online]. Available: https://www.mckinsey.com/featured-insights/the-rise-of-quantum-computing

[3] S. Wang, Z. Pei, C. Wang, and J. Wu, "Shaping the future of the application of quantum computing in intelligent transportation system," *Intell. Converged Netw.*, vol. 2, no. 4, pp. 259–276, Dec. 2021.

[4] V. V. Dixit and C. Niu, "Quantum computing for transport network design problems," *Sci. Rep.*, vol. 13, no. 1, p. 12267, 2023.

[5] X. Liu, J. Xu, K. Zheng, G. Zhang, J. Liu, and N. Shiratori, "Throughput maximization with an AoI constraint in energy harvesting D2D-enabled cellular networks: An MSRA-TD3 approach," *IEEE Trans. Wireless Commun.*, vol. 24, no. 2, pp. 1448–1466, Feb. 2025, doi: 10.1109/TWC.2024.3509475.

[6] K. Zheng, R. Luo, X. Liu, J. Qiu, and J. Liu, "Distributed DDPG-based resource allocation for age of information minimization in mobile wireless-powered Internet of Things," *IEEE Internet Things J.*, vol. 11, no. 17, pp. 29102–29115, Sep. 2024, doi: 10.1109/JIOT.2024.3406044.

[7] N. A. Khalek, D. H. Tashman, and W. Hamouda, "Advances in machine learning-driven cognitive radio for wireless networks: A survey," *IEEE Commun. Surveys Tuts.*, vol. 26, no. 2, pp. 1201–1237, 2nd Quart., 2024, doi: 10.1109/COMST.2023.3345796.

[8] R. Pal, N. Gupta, A. Prakash, R. Tripathi, and J. J. P. C. Rodrigues, "Deep reinforcement learning based optimal channel selection for cognitive radio vehicular ad-hoc network," *IET Commun.*, vol. 14, no. 19, pp. 3464–3471, Dec. 2020.

[9] X. Liu, C. Sun, M. Zhou, B. Lin, and Y. Lim, "Reinforcement learning based dynamic spectrum access in cognitive Internet of Vehicles," *China Commun.*, vol. 18, no. 7, pp. 58–68, Jul. 2021.

[10] H. Xie, R. Lin, J. Wang, M. Zhang, and C. Cheng, "Power allocation of energy harvesting cognitive radio based on deep reinforcement learning," in *Proc. 5th Int. Conf. Commun. Inf. Syst. (ICCIS)*, Oct. 2021, pp. 45–49.

[11] D. H. Tashman, S. Cherkaoui, and W. Hamouda, "Performance optimization of energy-harvesting underlay cognitive radio networks using reinforcement learning," in *Proc. Int. Wireless Commun. Mobile Comput. (IWCMC)*, Jun. 2023, pp. 1160–1165.

[12] M. Chen, U. Challita, W. Saad, C. Yin, and M. Debbah, "Artificial neural networks-based machine learning for wireless networks: A tutorial," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 4, pp. 3039–3071, 4th Quart., 2019.

[13] W. Zhao, T. Weng, Y. Ruan, Z. Liu, X. Wu, X. Zheng, and N. Kato, "Quantum computing in wireless communications and networking: A tutorial-cum-survey," *IEEE Commun. Surveys Tuts.*, early access, Nov. 20, 2025, doi: 10.1109/COMST.2024.3502762.

[14] P. Botsinis, D. Alanis, Z. Babar, H. V. Nguyen, D. Chandra, S. X. Ng, and L. Hanzo, "Quantum search algorithms for wireless communications," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1209–1242, 2nd Quart., 2019.

[15] S. J. Nawaz, S. K. Sharma, S. Wyne, M. N. Patwary, and M. Asaduzzaman, "Quantum machine learning for 6G communication networks: State-of-the-Art and vision for the future," *IEEE Access*, vol. 7, pp. 46317–46350, 2019.

[16] T. Q. Duong, J. A. Ansere, B. Narottama, V. Sharma, O. A. Dobre, and H. Shin, "Quantum-inspired machine learning for 6G: Fundamentals, security, resource allocations, challenges, and future research directions," *IEEE Open J. Veh. Technol.*, vol. 3, pp. 375–387, 2022.

[17] B. Narottama, Z. Mohamed, and S. Aïssa, "Quantum machine learning for next-G wireless communications: Fundamentals and the path ahead," *IEEE Open J. Commun. Soc.*, vol. 4, pp. 2204–2224, 2023.

[18] B. Narottama and S. Y. Shin, "Quantum neural networks for resource allocation in wireless communications," *IEEE Trans. Wireless Commun.*, vol. 21, no. 2, pp. 1103–1116, Feb. 2022, doi: 10.1109/TWC.2021.3102139.

[19] F. Zaman, A. Farooq, M. A. Ullah, H. Jung, H. Shin, and M. Z. Win, "Quantum machine intelligence for 6G URLLC," *IEEE Wireless Commun.*, vol. 30, no. 2, pp. 22–30, Apr. 2023.

[20] A. Singh, K. Dev, H. Siljak, H. D. Joshi, and M. Magarini, "Quantum internet—Applications, functionalities, enabling technologies, challenges, and research directions," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 4, pp. 2218–2247, 4th Quart., 2021.

[21] Z. Li, K. Xue, J. Li, L. Chen, R. Li, Z. Wang, N. Yu, D. S. L. Wei, Q. Sun, and J. Lu, "Entanglement-assisted quantum networks: Mechanics, enabling technologies, challenges, and research directions," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 4, pp. 2133–2189, 4th Quart., 2023.

[22] Z. Yang, M. Zolanvari, and R. Jain, "A survey of important issues in quantum computing and communications," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 2, pp. 1059–1094, Feb. 2023.

[23] J. L. O'Brien, "Optical quantum computing," *Science*, vol. 318, no. 5856, pp. 1567–1570, Dec. 2007.

[24] Q. A. Memon, M. Al Ahmad, and M. Pecht, "Quantum computing: Navigating the future of computation, challenges, and technological breakthroughs," *Quantum Rep.*, vol. 6, no. 4, pp. 627–663, Nov. 2024, doi: 10.3390/quantum6040039.

[25] D. S. Srivastava, D. M. Malik, and S., "Advances in quantum computing: A comprehensive review of quantum algorithms and applications in physics," *Int. J. Phys. Math. Sci.*, vol. 5, no. 1, pp. 50–56, Jul. 2023. [Online]. Available: https://www.physicsjournal.net/archives/2023.v5.i1.A.60/advances-in-quantum-computing-a-comprehensive-review-of-quantum-algorithms-and-applications-in-physics

[26] R. Mandelbaum, A. Corcoles, and J. Gambetta. *IBM's Big Bet on the Quantum-Centric Supercomputer*. Accessed: Sep. 19, 2024. [Online]. Available: https://spectrum.ieee.org/ibm-quantum-computer-2668978269

[27] S. K. Sood and Pooja, "Quantum computing review: A decade of research," *IEEE Trans. Eng. Manag.*, vol. 71, pp. 6662–6676, 2023, doi: 10.1109/TEM.2023.3284689.

[28] M. Settembre, "A 5G core network challenge: Combining flexibility and security," in *Proc. AEIT Int. Annu. Conf. (AEIT)*, Milan, Italy, Oct. 2021, pp. 1–6, doi: 10.23919/AEIT53387.2021.9627014.

[29] F. Arute et al., "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, Oct. 2019, doi: 10.1038/s41586-019-1666-5.

[30] C. Zhan and H. Gupta, "Quantum sensor network algorithms for transmitter localization," in *Proc. IEEE Int. Conf. Quantum Comput. Eng. (QCE)*, Bellevue, WA, USA, Sep. 2023, pp. 659–669, doi: 10.1109/QCE57702.2023.00081.

[31] E. Colella, L. Bastianelli, M. Khalily, F. Moglie, Z. Peng, and G. Gradoni, "Quantum optimisation of reconfigurable surfaces in complex propagation environments," in *Proc. 18th Eur. Conf. Antennas Propag. (EuCAP)*, Glasgow, U.K., Mar. 2024, pp. 1–5, doi: 10.23919/EUCAP60739.2024.10500955.

[32] S. Park, J. P. Kim, C. Park, S. Jung, and J. Kim, "Quantum multi-agent reinforcement learning for autonomous mobility cooperation," *IEEE Commun. Mag.*, vol. 62, no. 6, pp. 106–112, Jun. 2024.

[33] J. Prados-Garzon, T. Taleb, L. Chinchilla-Romero, and M. Shokrnezhad, "Deterministic 6GB-assisted quantum networks with slicing support: A new 6GB use case," *IEEE Netw.*, vol. 38, no. 1, pp. 87–95, Jan. 2024.

[34] B. B. Yousif and E. E. Elsayed, "Performance enhancement of an orbital-angular-momentum-multiplexed free-space optical link under atmospheric turbulence effects using spatial-mode multiplexing and hybrid diversity based on adaptive MIMO equalization," *IEEE Access*, vol. 7, pp. 84401–84412, 2019, doi: 10.1109/ACCESS.2019.2924531.

[35] J. L. Hevia, G. Peterssen, C. Ebert, and M. Piattini, "Quantum computing," *IEEE Softw.*, vol. 38, no. 5, pp. 7–15, Sep. 2021.

[36] A. M. Steane, "Error correcting codes in quantum theory," *Phys. Rev. Lett.*, vol. 77, no. 5, pp. 793–797, Jul. 1996, doi: 10.1103/PHYSREVLETT.77.793.

[37] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning," *Nature*, vol. 549, no. 7671, pp. 195–202, Sep. 2017, doi: 10.1038/nature23474.

[38] S. C. Kak, "Quantum neural computing," in *Advances in Imaging and Electron Physics*, vol. 94. Amsterdam, The Netherlands: Elsevier, 1995, pp. 259–313, doi: 10.1016/S1076-5670(08)70147-2.

[39] R. Chrisley, "Quantum learning," in *Proc. New Directions Cogn. Sci. Int. Symp.*, Aug. 1995, pp. 10–11. Accessed: Oct. 20, 2024. [Online]. Available: https://www.academia.edu/2623366/Quantum_learning

[40] S. Gupta and R. K. P. Zia, "Quantum neural networks," *J. Comput. Syst. Sci.*, vol. 63, no. 3, pp. 355–383, Nov. 2001, doi: 10.1006/jcss.2001.1769.

[41] Y. Kwak, W. J. Yun, S. Jung, and J. Kim, "Quantum neural networks: Concepts, applications, and challenges," in *Proc. 12th Int. Conf. Ubiquitous Future Netw. (ICUFN)*, Aug. 2021, pp. 413–416, doi: 10.1109/ICUFN49451.2021.9528698.

[42] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge, U.K.: Cambridge Univ. Press, 2010.

[43] M. M. Waldrop, "The chips are down for Moore's law," *Nature*, vol. 530, no. 7589, pp. 144–147, Feb. 2016, doi: 10.1038/530144a.

[44] P. A. M. Dirac, "The principles of quantum mechanics," in *International Series of Monographs on Physics*. London, U.K.: Oxford Univ. Press, 1982.

[45] J. J. Sakurai and J. Napolitano, "Modern quantum mechanics," in *Quantum Physics, Quantum Information and Quantum Computation*. Cambridge, U.K.: Cambridge Univ. Press, 2020, doi: 10.1017/9781108587280.

[46] W. J. Yun, Y. Kwak, J. P. Kim, H. Cho, S. Jung, J. Park, and J. Kim, "Quantum multi-agent reinforcement learning via variational quantum circuit design," in *Proc. IEEE 42nd Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2022, doi: 10.1109/ICDCS54860.2022.00151.

[47] R. Coelho, A. Sequeira, and L. Paulo Santos, "VQC-based reinforcement learning with data re-uploading: Performance and trainability," 2024, *arXiv:2401.11555*.

[48] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii, "Quantum circuit learning," *Phys. Rev. A, Gen. Phys.*, vol. 98, no. 3, Sep. 2018, Art. no. 032309, doi: 10.1103/PHYSREVA.98.032309.

[49] F. A. Cárdenas-López, L. Lamata, J. C. Retamal, and E. Solano, "Multiqubit and multilevel quantum reinforcement learning with quantum technologies," *PLoS ONE*, vol. 13, no. 7, Jul. 2018, Art. no. e0200455, doi: 10.1371/journal.pone.0200455.

[50] S. Oh, J. Choi, J.-K. Kim, and J. Kim, "Quantum convolutional neural network for resource-efficient image classification: A quantum random access memory (QRAM) approach," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Jan. 2021, pp. 50–52, doi: 10.1109/ICOIN50884.2021.9333906.

[51] J. Tian, X. Sun, Y. Du, S. Zhao, Q. Liu, K. Zhang, W. Yi, W. Huang, C. Wang, X. Wu, M.-H. Hsieh, T. Liu, W. Yang, and D. Tao, "Recent advances for quantum neural networks in generative learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 10, pp. 12321–12340, Oct. 2023, doi: 10.1109/TPAMI.2023.3272029.

[52] Q. C. Nguyen, L. B. Ho, L. Nguyen Tran, and H. Q. Nguyen, "Qsun: An open-source platform towards practical quantum machine learning applications," *Mach. Learn., Sci. Technol.*, vol. 3, no. 1, Mar. 2022, Art. no. 015034, doi: 10.1088/2632-2153/ac5997.

[53] M. Incudini, M. Grossi, A. Ceschini, A. Mandarino, M. Panella, S. Vallecorsa, and D. Windridge, "Resource saving via ensemble techniques for quantum neural networks," *Quantum Mach. Intell.*, vol. 5, no. 2, pp. 9–17, Sep. 2023, doi: 10.1007/s42484-023-00126-z.

[54] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, doi: 10.1038/nature14539.

[55] J. R. McClean, S. Boixo, V. N. Smelyanskiy, R. Babbush, and H. Neven, "Barren Plateaus in quantum neural network training landscapes," *Nature Commun.*, vol. 9, no. 1, p. 4812, Nov. 2018, doi: 10.1038/s41467-018-07090-4.

[56] M. Cerezo, A. Sone, T. Volkoff, L. Cincio, and P. J. Coles, "Cost function dependent barren Plateaus in shallow parametrized quantum circuits," *Nature Commun.*, vol. 12, no. 1, p. 1791, Mar. 2021, doi: 10.1038/s41467-021-21728-w.

[57] I. Cong, S. Choi, and M. D. Lukin, "Quantum convolutional neural networks," *Nature Phys.*, vol. 15, no. 12, pp. 1273–1278, Dec. 2019, doi: 10.1038/s41567-019-0648-8.

[58] S. Y. Chen, C. H. Yang, J. Qi, P.-Y. Chen, X. Ma, and H.-S. Goan, "Variational quantum circuits for deep reinforcement learning," *IEEE Access*, vol. 8, pp. 141007–141024, 2020, doi: 10.1109/ACCESS.2020.3010470.

[59] L.-H. Gong, J.-J. Pei, T.-F. Zhang, and N.-R. Zhou, "Quantum convolutional neural network based on variational quantum circuits," *Opt. Commun.*, vol. 550, Jan. 2024, Art. no. 129993, doi: 10.1016/j.optcom.2023.129993.

[60] Y. Song, J. Li, Y. Wu, S. Qin, Q. Wen, and F. Gao, "A resource-efficient quantum convolutional neural network," *Frontiers Phys.*, vol. 12, pp. 2–9, Apr. 2024, doi: 10.3389/fphy.2024.1362690.

[61] H. Wang, Z. Liang, J. Gu, Z. Li, Y. Ding, W. Jiang, Y. Shi, D. Pan, F. Chong, and S. Han, "TorchQuantum case study for robust quantum circuits," in *Proc. 41st IEEE/ACM Int. Conf. Comput.-Aided Design*. New York, NY, USA: ACM, Dec. 2022, pp. 1–9, doi: 10.1145/3508352.3561118.

[62] H. Wang, Y. Ding, J. Gu, Y. Lin, D. Z. Pan, F. T. Chong, and S. Han, "QuantumNAS: Noise-adaptive search for robust quantum circuits," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Apr. 2022, pp. 692–708, doi: 10.1109/HPCA53966.2022.00057.

[63] Y. Ma, R. Han, and W. Wang, "Prediction-based portfolio optimization models using deep neural networks," *IEEE Access*, vol. 8, pp. 115393–115405, 2020, doi: 10.1109/ACCESS.2020.3003819.

[64] Y. Assiri, "Stochastic optimization of plain convolutional neural networks with simple methods," 2020, *arXiv:2001.08856*.

[65] Q. Wang, Y. Ma, K. Zhao, and Y. Tian, "A comprehensive survey of loss functions in machine learning," *Ann. Data Sci.*, vol. 9, no. 2, pp. 187–212, Apr. 2022, doi: 10.1007/s40745-020-00253-5.

[66] L. Ciampiconi, A. Elwood, M. Leonardi, A. Mohamed, and A. Rozza, "A survey and taxonomy of loss functions in machine learning," 2023, *arXiv:2301.05579*.

[67] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[68] R. M. Schmidt, F. Schneider, and P. Hennig, "Descending through a crowded valley-benchmarking deep learning optimizers," in *Proc. 38th Int. Conf. Mach. Learning*, Jul. 2021, pp. 9367–9376. Accessed: Oct. 6, 2024. [Online]. Available: https://proceedings.mlr.press/v139/schmidt21a.html

[69] J. Lin, H. Yin, W. Ping, P. Molchanov, M. Shoeybi, and S. Han, "VILA: On pre-training for visual language models," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2024, pp. 26679–26689, doi: 10.1109/CVPR52733.2024.02520.

[70] W. R. W. Mohd and L. Abdullah, "Similarity measures of Pythagorean fuzzy sets based on combination of cosine similarity measure and Euclidean distance measure," *AIP Conf. Proc.*, vol. 2018, pp. 1–5, Jun. 2018, Art. no. 030017, doi: 10.1063/1.5041661.

[71] T. Srikaewsiew, K. Khianchainat, A. Tharatipyakul, S. Pongnumkul, and S. Kanjanawattana, "A comparison of the instructor-trainee dance dataset using cosine similarity, Euclidean distance, and angular difference," in *Proc. 26th Int. Comput. Sci. Eng. Conf. (ICSEC)*, Dec. 2022, pp. 235–240, doi: 10.1109/ICSEC56337.2022.10049368.

[72] M. R. da Silva, O. A. de Carvalho, R. F. Guimarães, R. A. T. Gomes, and C. R. Silva, "Wheat planted area detection from the MODIS NDVI time series classification using the nearest neighbour method calculated by the Euclidean distance and cosine similarity measures," *Geocarto Int.*, vol. 35, no. 13, pp. 1400–1414, Oct. 2020, doi: 10.1080/10106049.2019.1581266.

[73] C. X. Wang and W. P. Tay, "Semi-nonparametric estimation of distribution divergence in non-Euclidean spaces," 2022, *arXiv:2204.02031*.

[74] P.-E. Danielsson, "Euclidean distance mapping," *Comput. Graph. Image Process.*, vol. 14, no. 3, pp. 227–248, Nov. 1980, doi: 10.1016/0146-664X(80)90054-4.

[75] X. Peng, H. Liu, and X. Zhu, "A study on similarity and difficulty evaluation of elementary school mathematics application problems based on cosine similarity and AHP," *Curriculum Teach. Methodol.*, vol. 6, no. 20, pp. 108–115, Nov. 2023, doi: 10.23977/curtm.2023.062017.

[76] R. T. Hassan and N. S. Ahmed, "Evaluating of efficacy semantic similarity methods for comparison of academic thesis and dissertation texts," *Sci. J. Univ. Zakho*, vol. 11, no. 3, pp. 396–402, Aug. 2023, doi: 10.25271/sjuoz.2023.11.3.1120.

[77] *Wireless LANs, IEEE Standards Association*, IEEE Standard 802.11, 2023. [Online]. Available: https://standards.ieee.org/ieee/802.11/7

[78] Bluetooth SIG. *Specification of the Bluetooth System*. Accessed: Oct. 2023. [Online]. Available: https://www.bluetooth.com/specifications/specs/

[79] *IEEE Standard for Low-Rate Wireless Networks*, IEEE IEEE Standard 802.15.4-2020, 2023. [Online]. Available: https://standards.ieee.org/ieee/802.15.4/7029/.

[80] LoRa Alliance. *LoRaWAN Specification V1.1*. Accessed: Oct. 2023. [Online]. Available: https://resources.lora-alliance.org/technical-specifications/lorawan-specification-v1-1.

[81] ETSI. *DECT—Digital Enhanced Cordless Telecommunications*. Accessed: Oct. 2023. [Online]. Available: https://www.etsi.org/technologies/dect

[82] A. S. Tanenbaum and D. Wetherall, *Computer Networks*. London, U.K.: Pearson, 2010.

[83] S. Siddiqui, J. Holzer, and J. Malcarne. (2024). *Do-It-Yourself Quantum Neural Networks for Wireless Comunication Optimization*. [Online]. Available: https://github.com/TREXJET/DIY_QNN_Wireless_Paper

[84] A. Ahmadkhaniha and M. Bathaee, "Noisy intermediate-scale quantum computer-compatible error correction of quantum data using modified dissipative quantum neural networks," *Phys. Rev. A, Gen. Phys.*, vol. 109, no. 3, Mar. 2024, Art. no. 032620, doi: 10.1103/physreva.109.032620.
[85] M. Kashif and S. Al-Kuwari, "The impact of cost function globality and locality in hybrid quantum neural networks on NISQ devices," *Mach. Learning: Sci. Technol.*, vol. 4, no. 1, Mar. 2023, Art. no. 015004, doi: 10.1088/2632-2153/acb12f.
[86] A. Furutanpey, J. Barzen, M. Bechtold, S. Dustdar, F. Leymann, P. Raith, and F. Truger, "Architectural vision for quantum computing in the edge-cloud continuum," in *Proc. IEEE Int. Conf. Quantum Softw. (QSW)*, Jul. 2023, pp. 88–103, doi: 10.1109/QSW59989.2023.00021.
[87] Y. Kawase, "Distributed quantum neural networks via partitioned features encoding," *Quantum Mach. Intell.*, vol. 6, no. 1, p. 15, Mar. 2024, doi: 10.1007/s42484-024-00153-4.
[88] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. 33rd Int. Conf. Neural Inf. Process. Syst.*, Red Hook, NY, USA: Curran Associates, 2019, pp. 8026–8037.
[89] *Cirq Developers*, Zenodo, Google Quantum AI, Mountain View, CA, USA, May 29, 2024.
[90] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross, B. R. Johnson, and J. M. Gambetta, "Quantum computing with qiskit," 2024, *arXiv:2405.08810*.
[91] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. 32nd Int. Conf. Mach. Learn.*, 2015, pp. 448–456.
[92] S. G. K. Patro and K. K. Sahu, "Normalization: A preprocessing stage," 2015, *arXiv:1503.06462*.
[93] S. Endo, Z. Cai, S. C. Benjamin, and X. Yuan, "Hybrid quantum-classical algorithms and quantum error mitigation," *J. Phys. Soc. Jpn.*, vol. 90, no. 3, Mar. 2021, Art. no. 032001, doi: 10.7566/jpsj.90.032001.
[94] A. Abbas et al., "Challenges and opportunities in quantum optimization," 2023, *arXiv:2312.02279*.
[95] A. B. Finnila, M. A. Gomez, C. Sebenik, C. Stenson, and J. D. Doll, "Quantum annealing: A new method for minimizing multidimensional functions," *Chem. Phys. Lett.*, vol. 219, nos. 5–6, pp. 343–348, Mar. 1994, doi: 10.1016/0009-2614(94)00117-0.
[96] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross, B. R. Johnson, and J. M. Gambetta, "Quantum computing with Qiskit," 2024, *arXiv:2405.08810*.
[97] V. Bergholm et al., "PennyLane: Automatic differentiation of hybrid quantum-classical computations," 2018, *arXiv:1811.04968*.

SALIK SIDDIQUI (Student Member, IEEE) received the B.S. and M.S. degrees in electrical and computer engineering from Worcester Polytechnic Institute, in 2014 and 2024, respectively. He is currently an Engineer with NVIDIA, specializing in artificial intelligence. His research interests include artificial intelligence for engineering systems, applied quantum computing, quantum communication, wireless communication networks in space, and distributed systems.

JUSTIN HOLZER (Student Member, IEEE) received the B.S. degree in electrical engineering from The University of Utah, Salt Lake City, UT, USA, in 2001, the M.S. degree in electrical engineering from Brigham Young University, Provo, UT, USA, in 2004, and the Master of Business Administration (M.B.A.) degree from the University of California at Davis, Davis, CA, USA, in 2007. He is currently pursuing the Ph.D. degree in electrical and computer engineering with Worcester Polytechnic Institute (WPI), Worcester, MA, USA. In 2015, he joined the MIT Lincoln Laboratory, Lexington, MA, USA, as a Technical Staff. His research interests include wireless communications, signal processing, cellular communications, machine learning, and quantum computing applications.

JOSHUA MALCARNE (Student Member, IEEE) received the B.S. and M.S. degrees in computer science from WPI. He is a member with the WPI Wireless Innovation Laboratory, with research interests focused on machine learning and distributed computing. He has also completed work in pulse compression, signal design, and related domains during his academic career.

GALAHAD M. B. WERNSING received the B.S., M.S., and Ph.D. degrees in electrical and computer engineering from Worcester Polytechnic Institute, in 2019, 2020, and 2024, respectively. During the Covid lockdowns, they started Holy Grail Laboratories, a company focused on upgrading vintage automotive electronics and has since grown the company to be a global leader in fuse box modernization. Their research interests include wireless communications, novel radar systems, low-level computer security, and analog circuits.

ALEXANDER M. WYGLINSKI (Senior Member, IEEE) received the B.Eng. and Ph.D. degrees in electrical engineering from McGill University, Montreal, Canada, in 1999 and 2005, respectively, and the M.Sc.(Eng.) degree in electrical engineering from Queen's University, Kingston, Canada, in 2000. He is currently the Associate Dean of Graduate Studies, a Professor of electrical engineering and robotics engineering, and the Director of the Wireless Innovation Laboratory, Worcester Polytechnic Institute (WPI), Worcester, MA, USA. His current research interests include wireless communications, cognitive radio, machine learning for wireless systems, software-defined radio prototyping, connected and autonomous vehicles, and dynamic spectrum sensing, characterization, and access. He served as the President of the IEEE Vehicular Technology Society, from 2018 to 2019, as well as a Technical Editor for *IEEE Communications Magazine*, from 2011 to 2021.