PristiQ: A Co-Design Framework for Preserving Data Security of Quantum Learning in the Cloud

Zhepeng Wang[†, §], Yi Sheng[†], Nirajan Koirala[‡], Kanad Basu[*], Taeho Jung[‡], Cheng-Chang Lu[¶], Weiwen Jiang[†, §]
[†]Department of Electrical and Computer Engineering, George Mason University.
[§]Quantum Science and Engineering Center, George Mason University.
[‡]Department of Computer Science and Engineering, University of Notre Dame.
[*]Department of Electrical & Computer Engineering, University of Texas at Dallas
[¶]Qradle Inc.
{zwang48, wjiang8}@gmu.edu

2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI) | 979-8-3503-5411-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/ISVLSI61997.2024.00079

***Abstract*—Benefiting from cloud computing, today's early-stage quantum computers can be remotely accessed via the cloud services, known as Quantum-as-a-Service (QaaS). However, it poses a high risk of data leakage in quantum machine learning (QML). To run a QML model with QaaS, users need to locally compile their quantum circuits including the subcircuit of data encoding first and then send the compiled circuit to the QaaS provider for execution. If the QaaS provider is untrustworthy, the subcircuit to encode the raw data can be easily stolen. Therefore, we propose a co-design framework for Preserving the data security of QML with the QaaS paradigm, namely PristiQ. By introducing an encryption subcircuit with extra secure qubits associated with a user-defined security key, the security of data can be greatly enhanced. And an automatic search algorithm is proposed to optimize the model to maintain its performance on the encrypted quantum data. Experimental results on simulation and the actual IBM quantum computer both prove the ability of PristiQ to provide high security for the quantum data while maintaining the model performance in QML.**

***Index Terms*—Cloud Quantum Computing, Quantum Machine Learning, Quantum Data Security**

I. Introduction

Over decades of development in quantum science, we are now witnessing the breakthrough of technologies in designing quantum computers: the number of qubits in the actual quantum computers has increased rapidly, from 5 qubits in 2016 to 433 qubits in 2022 [5]. With access to these actual quantum computers via the cloud services, known as Quantum-as-a-Service (QaaS), various studies have been done to explore the quantum advantage in different problems [8], [11], where quantum machine learning (QML), in particular, quantum neural network (QNN) has attracted great research efforts recently [6], [9].

Although QNN is promising in many real applications, the current QaaS paradigm for the execution of QNN poses a high risk of data leakage due to the raw data delivery to the cloud. Fig. 1 shows a typical threat model in the QaaS paradigm for QML applications. In this case, the local user and the attacker may train their own QNN models on the same data but target different tasks. For example, with the same medical images, the user could build a QNN for classification while the attacker could build a different QNN for segmentation. The training process of a QNN is usually executed in a hybrid quantum-classical way. For instance, the classical data $\boldsymbol{Y_1}$ will first be encoded as a quantum circuit $D(\boldsymbol{Y_1})$ with a data encoding method (e.g., amplitude encoding). $D(\boldsymbol{Y_1})$ and the quantum circuit $QC(\boldsymbol{\theta})$ for the QNN will be provided to the local quantum compiler, which will generate a compiled quantum circuit $QC(\boldsymbol{\theta}) \cdot D(\boldsymbol{Y_1})$. The compiled circuit is then sent to the quantum cloud provider to run the forward pass. And the forward results will be sent back to the user to compute the gradients of learnable parameters $\boldsymbol{\theta}$ through parameter shift [7]. For the quantum circuit $QC(\boldsymbol{\omega})$ of the attacker, it can be trained in the same way.

After the training process, the compiled circuit $QC(\boldsymbol{\theta}) \cdot D(\boldsymbol{X})$ will be sent to the quantum cloud provider for inference to get an accurate classification of sensitive user data $\boldsymbol{X}$. However, if the quantum cloud provider is untrustworthy, the attacker can directly get access to $QC(\boldsymbol{\theta}) \cdot D(\boldsymbol{X})$. The encoding quantum circuit $D(\boldsymbol{X})$ can be decoupled from $QC(\boldsymbol{\theta}) \cdot D(\boldsymbol{X})$ since the data encoding algorithms have fixed circuit patterns that are easy to be detected. Moreover, with $QC(\boldsymbol{\omega})$, an accurate segmentation of $D(\boldsymbol{X}$ can be made, which means that the attacker can obtain more information from $D(\boldsymbol{X})$ with data analysis methods. Such kind of data leakage can not only cause privacy breaches to the users but also bring huge damage to the reputation of the institution that leverages the power of QNNs for their applications. Therefore, the preservation of data security is of great importance in QML applications.

There exist many works on blind quantum computing [1], [3], which focus on the security of quantum circuits. However, they assume that the user has a lightweight quantum terminal to preprocess the data in quantum states, which does not hold in the current QaaS paradigm for quantum applications. Traditional secure computation methods like homomorphic encryption [4] cannot be directly applied to quantum computing since the involved complicated computing operations cannot be implemented by quantum operations. Therefore, a quantum computing-based methodology is necessary to protect the raw input data to the QNNs in the context of QaaS.

Pristiq is a prescription medicine to treat depression. We envision our framework to treat the depression by data leakage in future quantum machine learning research.

2159-3477/24/$31.00 ©2024 IEEE
DOI 10.1109/ISVLSI61997.2024.00079

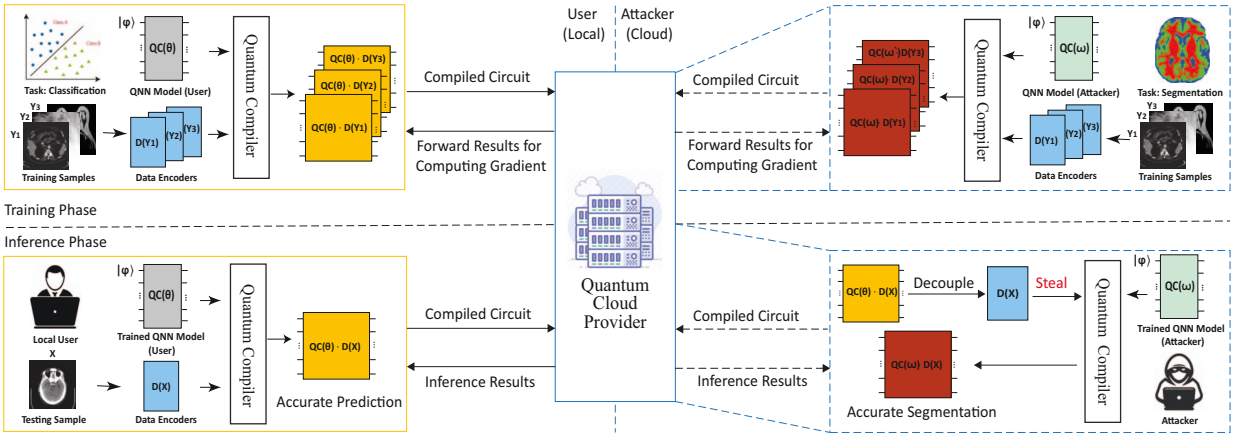


Fig. 1. Illustration of the typical threat model in QaaS paradigm for QML applications: The attacker in the untrustworthy cloud quantum provider can steal the user data for its own QML task.

In this paper, we make the very first attempt to preserve data security in QML applications by formulating a circuit-compiler-model cross-layer design framework, namely PristiQ. The proposed framework is composed of three components: *i) PriCircuit* incorporates an encryption subcircuit into the subcircuit of data encoding to keep data security; *ii) PriCompiler* obfuscates the encryption subcircuit; and *iii) PriModel* automatically searches for the optimal design of the QNN model in order to maintain the performance when directly conducting computations on the encrypted data. These components will work together to provide data security while maintaining high performance for the QML tasks through QaaS.

The main contributions of this paper are as follows.

- The first and the main contribution of this paper is the proposal of PristiQ, which is, to the best of our knowledge, the first framework to secure input data to QNNs under QaaS paradigm.
- PristiQ provides a fundamental understanding of the design of a secure quantum computing system: a cross-layer co-design is required to preserve data security in quantum computing. This design philosophy is not limited to QML applications but can also be applied to other types of quantum applications.
- PristiQ brings the concept of model adaptation to the design of QNN, which preserves high performance on the encrypted data.

II. PristiQ Framework

Fig. 2 shows an overview of PristiQ. Given the data encoding subcircuit $D(\boldsymbol{X})$ and QNN subcircuit $QC(\boldsymbol{\theta})$ as inputs, which is shown in Fig. 2 (a), PristiQ will sequentially go through the three components: *PriCircuit* in Fig. 2(b), *PriCompiler* in Fig. 2(c) and *PriModel* in Fig. 2(d).

Based on $D(\boldsymbol{X})$ and $QC(\boldsymbol{\theta})$, *PriCircuit* will reconstruct the original circuit $QC(\boldsymbol{\theta}) \cdot D(\boldsymbol{X})$ by inserting an encryption subcircuit $E(\boldsymbol{\delta})$ between $D(\boldsymbol{X})$ and $QC(\boldsymbol{\theta})$, which introduces extra qubits, as shown in Fig. 2(b). To avoid the attacker detecting the boundary between $D(\boldsymbol{X})$ and $E(\boldsymbol{\delta})$ easily, *PriCompiler* is designed to obfuscate $D(\boldsymbol{X})$ and $E(\boldsymbol{\delta})$. More specifically, it utilizes the compiling optimization to form a merged circuit $E(\boldsymbol{\delta}) \cdot D(\boldsymbol{X})$, making their boundary indistinguishable. In this way, the attacker could only extract the encrypted data encoded by $E(\boldsymbol{\delta}) \cdot D(\boldsymbol{X})$ instead of the raw input data encoded by $D(\boldsymbol{X})$. However, directly applying the original QNN subcircuit $QC(\boldsymbol{\theta})$ to the encrypted data may incur significant performance degradation. Therefore, to maintain the performance on encrypted data, *PriModel* will revise $QC(\boldsymbol{\theta})$ by replacing it with subcircuit $QC'(\boldsymbol{\theta'})$. The neural architecture of the QNN for $QC'(\boldsymbol{\theta'})$ is designed by using an automatic search engine based on reinforcement learning, aiming at the best performance on the encrypted data. In the following content of this section, we will introduce the details of all three components in PristiQ.

A. PriCircuit

Design Principle: The main purpose of *PriCircuit* is to provide data security, i.e., the raw input data should be hidden; however, the encrypted data should also preserve the critical information contained by the raw input data, such that the encrypted data can still be learned effectively. Unlike conventional neural networks in classical computing, most quantum neural networks [2], [6] do not extract the spatial features. Therefore, the most important information in the raw input data is the relative relationship among its features. Based on this observation, we propose a two-stage encryption process for *PriCircuit*. And the details are as follows.

Design overview: Fig. 3 shows the detailed design of encryption subcircuit $E(\boldsymbol{\delta})$ generated by *PriCircuit* with an example. As shown in Fig. 3(a), there are N qubits used to encode raw input data $\boldsymbol{X}$ with 2^N features, named data qubits. And M extra qubits are added to the N data qubits, called secure qubits. The first stage applies the subcircuit $S(\boldsymbol{\delta})$, which is a part of $E(\boldsymbol{\delta})$, to operate on the newly added M secure qubits with random parameter set $\boldsymbol{\delta}$ to perform amortization on the amplitudes from data qubits to secure qubits; then, the second stage utilizes the remaining part of $E(\boldsymbol{\delta})$ (i.e., subcircuit P) to perform permutation on the amplitudes of the quantum state processed by $S(\boldsymbol{\delta})$ since the permutation matrix can be implemented by quantum circuits. Based on the

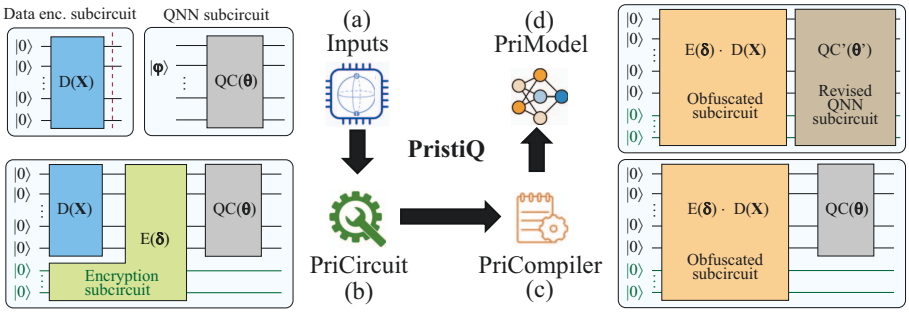


Fig. 2. Illustration of PristiQ framework: (a) the data encoding subcircuit $D(\boldsymbol{X})$ and QNN subcircuit $QC(\boldsymbol{\theta})$ as the inputs; (b) *PriCircuit*, adding the encryption subcircuit $E(\boldsymbol{\delta})$; (c) *PriCompiler*, obfuscating $D(\boldsymbol{X})$ and $E(\boldsymbol{\delta})$ by forming $E(\boldsymbol{\delta}) \cdot D(\boldsymbol{X})$; (d) *PriModel*, revising QNN subcircuit $QC(\boldsymbol{\theta})$ to $QC'(\boldsymbol{\theta'})$ for maintaining the performance on encrypted data.

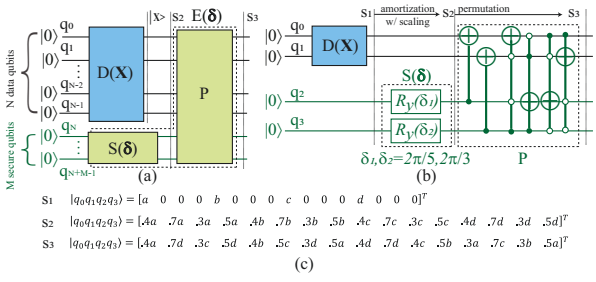


Fig. 3. *PriCircuit*: (a) encryption subcircuit $E(\boldsymbol{\delta})$ composed of $S(\boldsymbol{\delta})$ and P; (b) an example of *PriCircuit* with 2 data qubits and 2 secure qubits; (c) transformations of quantum states: $S_1 \rightarrow S_2$ and $S_2 \rightarrow S_3$.

design of *PriCircuit*, we define the security key in PristiQ as a tuple $\langle \boldsymbol{\delta}, P \rangle$, where $\boldsymbol{\delta}$ is the random parameter set used in the amortization with scaling while P is the design of random permutation circuit.

Stage 1: amortization w/ scaling. As shown in Fig. 3(a), stage 1 starts with initial quantum states S_1, where $S_1 = |\boldsymbol{X}\rangle \otimes |0\rangle^{\otimes M} = |q_0 q_1 ... q_{N-1} q_N ... q_{N+M-1}\rangle$. Here $|\boldsymbol{X}\rangle$ is the encoded quantum state for the raw input data $\boldsymbol{X}$, using the data encoding subcircuit $D(\boldsymbol{X})$. Note that for S_1, only quantum basis state $|q_0 q_1 ... q_{N-1} 0...0\rangle$ has non-zero amplitudes since no quantum gates are applied to the M secure qubits. For example, in Fig. 3(b), the number of data qubits N is 2 while the number of secure qubits M is 2 as well. In this case, $S_1 = |q_0 q_1 q_2 q_3\rangle = |\boldsymbol{X}\rangle \otimes |00\rangle$ and only quantum state $|q_0 q_1 00\rangle$ has non-zero amplitudes, which is clearly shown in the state vector of S_1 in Fig. 3(c).

Stage 1 focuses on the transformation from state S_1 to state S_2, where each secure qubit is operated by Ry gate with a parameter that is randomly generated. More specifically, the k^{th} secure qubit is denoted as Sec_k and its initial state is $|0\rangle$. After applying an Ry gate with a parameter δ_k to it, the state of Sec_k is converted to $[\cos\frac{\delta_k}{2}, \sin\frac{\delta_k}{2}]^T$, which is denoted by $|QS_k\rangle$. Therefore, the quantum state $|Q\rangle$, which is the state composed of all the secure qubits, can be computed as,

$$|Q\rangle = |QS_0\rangle \otimes |QS_1\rangle \otimes ... \otimes |QS_k\rangle \otimes ... \otimes |QS_{M-1}\rangle. \quad (1)$$

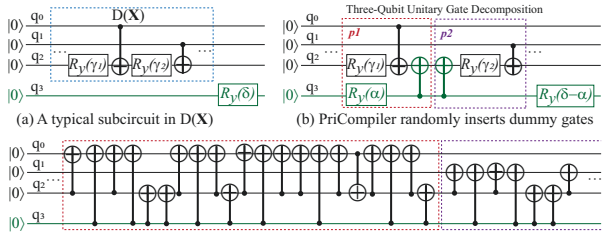
By combining the quantum states of data qubits $|\boldsymbol{X}\rangle$ and secure qubits $|Q\rangle$, we have $S_2 = |\boldsymbol{X}\rangle \otimes |Q\rangle$, where $|Q\rangle$ is served as the **scaling** vector with 2^M scaling factors for the **amortization** on the amplitudes from data qubits to secure qubits. For example, in Fig. 3(b), the amplitude on $|q_0 q_1 00\rangle$ is amortized to the other three states (i.e., $|q_0 q_1 01\rangle$, $|q_0 q_1 10\rangle$, $|q_0 q_1 11\rangle$) with scaling factors in $|Q\rangle$ determined by the parameter set $\boldsymbol{\delta}$. For instance, in Fig. 3(b), $|\boldsymbol{X}\rangle = [a, b, c, d]^T$. Since $\boldsymbol{\delta} = [\delta_0, \delta_1] = [\frac{\pi}{5}, \frac{\pi}{3}]$, we have $|QS_0\rangle = [cos\frac{\pi}{5}, sin\frac{\pi}{5}] = [0.81, 0.59]$ and $|QS_1\rangle = [cos\frac{\pi}{3}, sin\frac{\pi}{3}] = [0.5, 0.87]$. According to Equation (1), we have $|Q\rangle = [0.4, 0.7, 0.3, 0.5]$. We can then obtain $S_2 = [a \cdot |Q\rangle, b \cdot |Q\rangle, c \cdot |Q\rangle, d \cdot |Q\rangle]^T$. The specific value of S_2 is shown in Fig. 3(c). And there exist four groups of values in S_2, where the values in each group are multiplied with a unique amplitude from $|\boldsymbol{X}\rangle$. Since all the groups share the same scaling vector $|Q\rangle$, the relative magnitude between arbitrary pair of groups is equal to that between the unique amplitudes they multiply with. In this way, the relative relationship between features within raw input data $|\boldsymbol{X}\rangle$ is maintained.

Stage 2: permutation. Although the random parameter set $\boldsymbol{\delta}$ is unknown to the attacker, stage 1 is not sufficient for the encryption of data since the attacker could get $\boldsymbol{\delta}$ and $|\boldsymbol{X}\rangle$ by solving a system of trigonometric equations if they know the processing of stage 1 is applied. Therefore, *PriCircuit* further generates quantum state S_3 by randomly permuting the amplitudes in S_2 with a permutation matrix P. Since permutation matrix P is unknown to the attacker, the system of trigonometric equations cannot be formulated and thus not be solved. Note that because the permutation matrix is a unitary matrix, it is always feasible to implement it with a corresponding quantum circuit, as shown in Fig. 3(a). Therefore, we have $S_3 = P \cdot S_2$.. For the example in Fig. 3(b), it utilizes multiple CNOT gates to implement a permutation matrix and the specific value of S_3 after permutation is shown in Fig. 3(c). And it is obvious that there is no explicit group of values that shares the same multiplier after the permutation.

B. PriCompiler

Potential Threat and Design Principle: The main purpose of *PriCompiler* is to obfuscate $E(\boldsymbol{\delta})$. This is motivated by the fact that the boundary between the secure qubits and data qubits might be detected based on the design of *PriCircuit* since there are only Ry gates on the secure qubits in $S(\boldsymbol{\delta})$ while the data qubits on $D(\boldsymbol{X})$ contains multiple two-qubit gates. If the boundary is identified, then $D(\boldsymbol{X})$ can be extracted.

Therefore, we propose to introduce two-qubit gates on the secure qubits in $S(\boldsymbol{\delta})$, making the boundary between secure



(a) A typical subcircuit in D(**X**)
(b) PriCompiler randomly inserts dummy gates
(c) PriCompiler decomposes the blocks with three qubits based on their unitary matrices
(Note: Rz gates are removed for simplicity)

Fig. 4. *PriCompiler*: (a) a typical subcircuit in $D(\boldsymbol{X})$; (b) the subcircuit with dummy CNOT gates; (c) the compiled subcircuit of (b) using *PriCompiler*.

qubits and data qubits indistinguishable. However, naively adding more two-qubit gates may introduce extra undesired operations on the input data and thus disrupt the features for inference. Therefore, *PriCompiler* should try to avoid actual interaction with $D(\boldsymbol{X})$, which brings new challenges for obfuscation. Fortunately, by reviewing the QaaS paradigm in Fig. 1, we observe that the quantum compiler can be leveraged to make the obfuscation.

Design Details: *PriCompiler* follows a two-stage design flow: (1) a dummy gate insertion and (2) unitary gate decomposition. Based on the fact that two consecutive CNOT gates can be canceled out with each other (i.e., $I = CNOT \cdot CNOT$), we propose a method to create dummy gates and introduce two-qubit gates on the secure qubits. Fig. 4 shows an example of the whole design flow. The circuit in Fig. 4(a) is a typical subcircuit in $D(\boldsymbol{X})$. In this subcircuit, the three data qubits are operated with two-qubit gates while the single secure qubit only has an Ry gate with a random parameter δ. In Fig. 4(b), an Ry gate with two dummy CNOT gates is added to the secure qubit for obfuscation. More specifically, we first add an Ry gate with a randomly generated parameter α. We then adjust the parameter of the original Ry gate to $\delta - \alpha$ to ensure the functional correctness of the circuit. Between these two Ry gates, we add two consecutive CNOT gates. After this, the first stage is completed.

In the second stage, given the circuit with dummy CNOT gates from the first stage, the default quantum compiler will remove them. But in *PriCompiler*, it adds a barrier between the pair of dummy CNOT gates to split the circuit into two parts, i,e., p_1 and p_2, as shown in Fig. 4(b). In p_1 and p_2, each subcircuit will be assembled into a block if it includes exactly three qubits. In this example, p_1 and p_2 can be regarded as two blocks. Since all the gates in each block correspond to a unitary gate, the unitary matrix of each unitary gate can be calculated and further decomposed to a new circuit with different designs consisting of chosen basis gates. Following this rule, the resultant circuit compiled from the one in Fig. 4(b) is presented in Fig. 4(c). In Fig. 4(c), it is clearly shown that multiple two-qubit gates are introduced on the secure qubit q_3. Due to the limited space, we only keep the CNOT gate in the circuit in Fig. 4(c). And the consecutive CNOT gates can thus not be canceled out due to the existence of Rz gates between them.

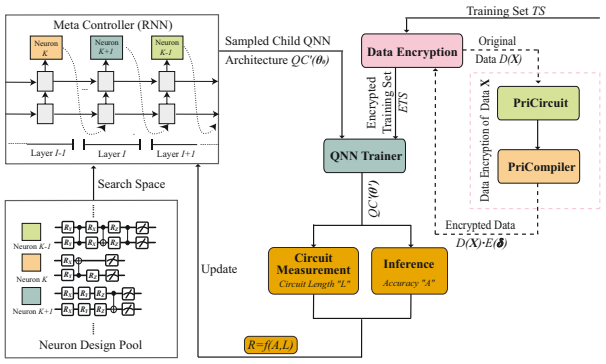


Fig. 5. The workflow of *PriModel*

C. PriModel

After the process of *PriCircuit* and *PriCompiler*, the total number of input qubits is increased from N to $N + M$. Therefore, the dimensions of features to the QNN model for processing is expanded from 2^N to 2^{N+M} as well. However, the original QNN subcircuit $QC(\boldsymbol{\theta})$ only contains operations on the N data qubits without consideration of M secure qubits, as shown in Fig. 2(c). Moreover, the original QNN subcircuit $QC(\boldsymbol{\theta})$ is designed and optimized for the raw input data instead of the encrypted data. As a result, directly applying this QNN model to the encrypted data can degrade the performance of QNN model, which is shown in Section III. Therefore, a reinforcement learning-based algorithm, namely *PriModel*, is proposed to automatically search for the best QNN architecture for the encrypted data.

Fig. 5 shows the overview of *PriModel*. It is composed of three components: (1) a meta controller to guide the whole process which is implemented with a recurrent neural network (RNN), (2) a neuron design pool to serve as the search space for the architecture of QNN, which consists of multiple quantum neuron designs. (3) a security-aware QNN evaluator (i.e., the right part of Fig. 5) to measure each sampled solution from the meta controller.

More specifically, for the meta controller, the RNN cell in time step I corresponds to the design of the quantum neuron in layer I of the QNN, which generates a probability distribution for the sampling from the neuron design pool. The sampled neuron design for each layer will be connected to form a child QNN $QC'(\boldsymbol{\theta_0})$, which is initialized with $\boldsymbol{\theta_0}$ and sent to the evaluator. During the evaluation, $QC'(\boldsymbol{\theta_0})$ will first be trained on the encrypted training set ETS, which is obtained by encrypting the data in the original training set TS. For each original data $D(\boldsymbol{X})$ in TS, it will go through *PriCircuit* and *PriCompiler* to get the encrypted data $D(\boldsymbol{X}) \cdot E(\boldsymbol{\delta})$. And the outcome will be saved in ETS. The trained QNN model $QC'(\boldsymbol{\theta'})$ is then evaluated to calculate two metrics, i.e., circuit length L and accuracy A. These two metrics are used for calculating the reward R to update the meta controller. The reward function is defined as,

$$R = A - b - \lambda \frac{L}{L_{base}}, \tag{2}$$

where b is the exponential moving average of the accuracies

TABLE I
EVALUATION OF PRICIRCUIT

Dataset	# Secure Qubits	Model Source	Accuracy (%)	PSNR(dB)
MNIST-2	0	User	99.06	34.28
		Attacker	99.06	
	1	Attacker	54.10 ± 16.06	19.35 ± 2.21
	2	Attacker	52.23 ± 6.23	18.21 ± 8.37
MNIST-3	0	User	92.56	34.82
		Attacker	92.29	
	1	Attacker	39.05 ± 6.12	20.00 ± 2.18
	2	Attacker	36.29 ± 8.90	18.69 ± 8.31
Fashion-2	0	User	87.19	26.06
		Attacker	86.54	
	1	Attacker	58.97 ± 17.33	18.15 ± 2.06
	2	Attacker	54.45 ± 13.64	16.7 ± 0.50
Fashion-3	0	User	77.41	25.32
		Attacker	77.38	
	1	Attacker	44.26 ± 10.86	17.81 ± 1.97
	2	Attacker	35.82 ± 10.48	17.02 ± 1.47

of previous sampled QNNs. L_{base} is a preset baseline circuit length for normalization. According to the available quantum computing resources, the user can use λ to control the trade-off between the accuracy and the complexity of circuit implementation for the sampled QNN.

III. EXPERIMENTAL RESULTS

A. Experimental Setting

Dataset. We evaluate PristiQ on MNIST-2 (class), -3 (class); Fashion-2 (class), -3 (class). When evaluated on the noiseless simulator, the data are downsampled to a resolution of 4×4 from 28×28, which needs 4 data qubits with amplitude encoding. When evaluated in noisy environments (i.e., noisy simulator of IBMQ Manila and IBMQ Manila), the data is downsampled to a resolution of 4×2 with 3 data qubits. In this case, due to the limited quantum resources, we use 100 samples and 150 samples from MNIST-2 and MNIST-3 for evaluation, respectively.

Security Keys. To generate the security keys, the range of random sampling of rotation angles is $[\frac{1\pi}{8}, \frac{7\pi}{8})$ and the permutation matrix P is implemented with random generation of CNOT gates.

Metrics. We use the peak signal-to-noise ratio (PSNR) to serve as a quantitative metric to evaluate the difference between two images. Since amplitude encoding requires normalization of the features within the original data, it introduces differences between the quantum data and the original data. Therefore, we computed the PSNR between the original data and the quantum data (i.e., the PSNR when the number of secure qubits is 0) to be the baseline for the comparison of PSNR.

QNNs. All the evaluated child QNNs are implemented with TorchQuantum [10]. They are then compiled and executed by Qiskit. Besides, for *PriModel*, there are 6 options [9] for quantum neural designs at each layer.

B. PriCircuit Effectively Protects Data

As Fig. 1 shows, we have two types of models for evaluation in this part, i.e., the model for user and the model for attacker.

TABLE II
EVALUATION OF PRISTIQ

Dataset	# Secure Qubits	Model Type	# Param	Circuit Length	Accuracy (%)
MNIST-2	0	User (Vanilia)	56	82	99.06
	1	User (Vanilia)	56	82	53.65 ± 14.98
		User (PristiQ)	**59.5**	**96.88**	**99.15 ± 0.05**
	2	User (Vanilia)	56	82	49.81 ± 4.79
		User (PristiQ)	**65**	**135**	**99.10 ± 0.04**
MNIST-3	0	User (Vanilia)	44	52	92.56
	1	User (Vanilia)	44	52	40.07 ± 9.52
		User (PristiQ)	**48.5**	**72.25**	**95.08 ± 0.27**
	2	User (Vanilia)	44	52	40.22 ± 9.96
		User (PristiQ)	**58.5**	**115.25**	**95.03 ± 0.24**
Fashion-2	0	User (Vanilia)	80	103	87.19
	1	User (Vanilia)	80	103	58.97 ± 17.35
		User (PristiQ)	**41.5**	**59.38**	**88.30 ± 0.19**
	2	User (Vanilia)	80	103	54.39 ± 13.54
		User (PristiQ)	**45.5**	**79.5**	**88.23 ± 0.16**
Fashion-3	0	User (Vanilia)	48	46	77.41
	1	User (Vanilia)	48	46	36.97 ± 9.26
		User (PristiQ)	**57.5**	**90.38**	**79.13 ± 0.09**
	2	User (Vanilia)	48	46	36.80 ± 13.42
		User (PristiQ)	**67**	**135.88**	**79.07 ± 0.10**

In general cases, the two models should target different types of tasks with the same dataset. But in this experiment, we trained two different models which both target the classification task for simplicity.

Table I reports the results for the evaluation of *PriCircuit*. It clearly shows that without encryption, the data stolen from the user can be utilized by the attacker with its own model effectively. More specifically, the performance of the model for attacker is close to that of the model for user and thus verifies the threat model in Fig. 1. For example, on MNIST-3, the accuracy of the model for attacker is 92.29 % while that of the model for user is 92.56 %.

When the data is encrypted by *PriCircuit*, the security of data can be preserved. It means that the model for the attacker performs badly on the encrypted data while the difference between the encrypted data and the original data is increased significantly. For example, with only 1 secure qubit, *PriCircuit* can reduce the accuracy and PSNR to 39.05 % and 20.0 dB on average on MNIST-3, respectively.

Moreover, increasing the number of secure qubits can further enhance the data security. For instance, on MNIST-3, the accuracy drops from 39.05 % to 36.29 % and the PSNR reduces from 20.0 dB to 18.69 dB respectively when increasing the secure qubits from 1 to 2.

C. PristiQ Can Enable Accurate Inference While Protecting Data

Table II shows the results for the evaluation of PristiQ on the noiseless simulator, where the model "User (Vanilla)" refers to the model for user optimized on the original data. Without encryption (i.e., the number of secure qubits is 0), it is clearly shown that only using *PriCircuit* to protect data will lead to the performance collapse of the model for user. For example, on MNIST-2, the accuracy drops from 99.06 % to 49.81 % on average with 2 secure qubits.

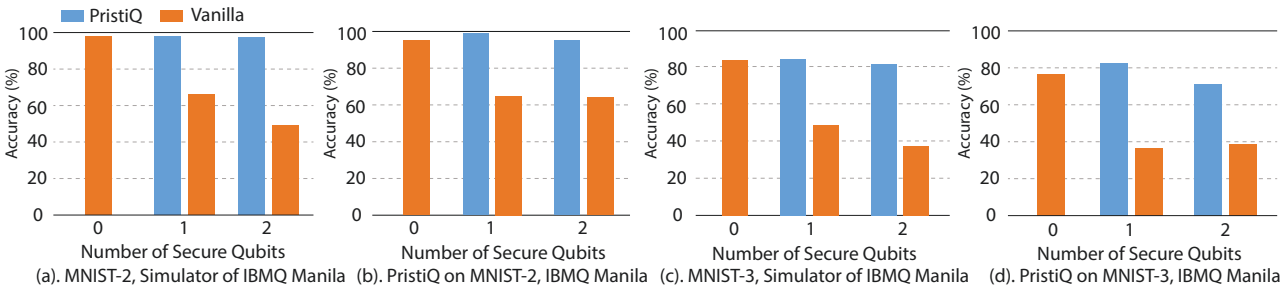


Fig. 6. Results of PristiQ for MNIST-2 on (a) simulator of IBMQ Manila and on (b) IBMQ Manila, and for MNIST-3 on (c) simulator of IBMQ Manila and on (d) IBMQ Manila.

By applying PristiQ to building the model, where *PriCircuit* is included, the performance of the model (i.e., "User (PristiQ)" in Table II) can be recovered on all the evaluated datasets. For example, on MNIST-2, the accuracy is recovered to 99.10% on average when encrypted with 2 secure qubits, which is even improved by 0.04 % compared with the model optimized on the original data.

D. Evaluation of PristiQ in Noisy Environments

Fig. 6 shows the effect of quantum noise on the performance of PristiQ on MNIST-2 and MNIST-3. In Fig. 6, the orange bar (Vanilla) denotes the model for user optimized on the original data while the blue bar (PristiQ) denotes the model optimized on the encrypted data with PristiQ. Fig. 6 (a) and (c) show the results on the noisy simulator of IBMQ Manila, while Fig. 6 (b) and (d) show the results on the actual quantum computer (i.e., IBMQ Manila).

For the vanilla model, we can conclude that its performance degrades significantly on the encrypted data generated by *PriCircuit* in noisy environments. For example, with 2 secure qubits, the accuracy decreases from 98 % to 49 % on the noisy simulator and drops from 95 % to 64 % on the actual quantum computer on MNIST-2.

By applying PristiQ to optimizing the model, the performance can still be recovered even with the quantum noise. For instance, with 2 secure qubits, the accuracy can be recovered from 49 % to 97 % on the noisy simulator and increased from 64 % to 95 % on the actual quantum computer on MNIST-2.

IV. Conclusion

In this paper, we made the very first exploration of the problem of data security for QML in the cloud. We proposed PristiQ, a framework to preserve data security in QML. By creating an encryption subcircuit with a user-defined security key, the important information in the original data is protected. Besides, PristiQ utilizes an automatic model optimizer to achieve high performance on the encrypted data. Extensive experiments in the noiseless quantum simulator and noisy quantum environments are conducted to show the effectiveness of PristiQ. Moreover, the design philosophy of PristiQ, a cross-layer co-design from the circuit level and compiler level to the application level, could be applied to guide the design of a secure quantum computing system in applications beyond QML.

Acknowledgment

This work is partly supported by the National Science Foundation (NSF) OAC-2311949 and OAC-2320957. The research used IBM Quantum resources via the Oak Ridge Leadership Computing Facility at the Oak Ridge National Lab, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. This project was also supported by resources provided by the Office of Research Computing at George Mason University (URL: https://orc.gmu.edu) and funded in part by grants from the National Science Foundation (Award Number 2018631). And we also thank Mason's QSEC and C-TASC centers for their support.

References

[1] A. Broadbent, J. Fitzsimons, and E. Kashefi, "Universal blind quantum computation," in *2009 50th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, 2009, pp. 517–526.

[2] Y. Cao, G. G. Guerreschi, and A. Aspuru-Guzik, "Quantum neuron: an elementary building block for machine learning on quantum computers," *arXiv preprint arXiv:1711.11240*, 2017.

[3] J. F. Fitzsimons, "Private quantum computation: an introduction to blind quantum computing and related protocols," *npj Quantum Information*, vol. 3, no. 1, pp. 1–11, 2017.

[4] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *International conference on machine learning*. PMLR, 2016, pp. 201–210.

[5] IBM, "Ibm unveils world's largest quantum computer at 433 qubits," *https://www.newscientist.com/article/2346074-ibm-unveils-worlds-largest-quantum-computer-at-433-qubits*, 2022, accessed: November, 2022.

[6] W. Jiang, J. Xiong, and Y. Shi, "A co-design framework of neural networks and quantum circuits towards quantum advantage," *Nature communications*, vol. 12, no. 1, pp. 1–13, 2021.

[7] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii, "Quantum circuit learning," *Physical Review A*, vol. 98, no. 3, p. 032309, 2018.

[8] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[9] S. Sim, P. D. Johnson, and A. Aspuru-Guzik, "Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms," *Advanced Quantum Technologies*, vol. 2, no. 12, p. 1900070, 2019.

[10] H. Wang, Y. Ding, J. Gu, Z. Li, Y. Lin, D. Z. Pan, F. T. Chong, and S. Han, "Quantumnas: Noise-adaptive search for robust quantum circuits," in *The 28th IEEE International Symposium on High-Performance Computer Architecture (HPCA-28)*, 2022.

[11] H.-S. Zhong, H. Wang, Y.-H. Deng, M.-C. Chen, L.-C. Peng, Y.-H. Luo, J. Qin, D. Wu, X. Ding, Y. Hu *et al.*, "Quantum computational advantage using photons," *Science*, 2020.