Benchmarking simulated and physical quantum processing units using quantum and hybrid algorithms

Mohammad Kordzanganeh, Markus Buchberger, Maxim Povolotskii, Wilhelm Fischer, Andrii Kurkin, Wilfrid Somogyi, Asel Sagingalieva, Markus Pflitsch, and Alexey Melnikov
Terra Quantum AG, 9000 St. Gallen, Switzerland and
QMware AG, 9000 St. Gallen, Switzerland
(Dated: November 29, 2022)

Powerful hardware services and software libraries are vital tools for quickly and affordably designing, testing, and executing quantum algorithms. A robust large-scale study of how the performance of these platforms scales with the number of qubits is key to providing quantum solutions to challenging industry problems. Such an evaluation is difficult owing to the availability and price of physical quantum processing units. This work benchmarks the runtime and accuracy for a representative sample of specialized high-performance simulated and physical quantum processing units. Results show the QMware cloud computing service can reduce the runtime for executing a quantum circuit by up to 78% compared to the next fastest option for algorithms with fewer than 27 qubits. The AWS SV1 simulator offers a runtime advantage for larger circuits, up to the maximum 34 qubits available with SV1. Beyond this limit, QMware provides the ability to execute circuits as large as 40 qubits. Physical quantum devices, such as Rigetti's Aspen-M2, can provide an exponential runtime advantage for circuits with more than 30. However, the high financial cost of physical quantum processing units presents a serious barrier to practical use. Moreover, of the four quantum devices tested, only IonQ's Harmony achieves high fidelity with more than four qubits. This study paves the way to understanding the optimal combination of available software and hardware for executing practical quantum algorithms.

I. INTRODUCTION

Quantum computing is a rapidly growing field of technology with increasingly useful applications across both industry and research. This new paradigm of computing has the potential to solve classically-intractable problems, by exploiting an exponentially-increasing computational space. This allows quantum algorithms to dramatically reduce the runtime for solving computationally resource-intensive problems.

There is a plethora of quantum algorithms, of which parameterized quantum circuits represent the most general form. Quantum neural networks (QNNs) leverage powerful techniques developed for classical neural networks, to optimize this parameterized structure, and have already been applied to solve a number of industrial problems [1–4]. The complexity and performance of classical neural networks employed to solve data-intensive problems has grown dramatically in the last decade. Although algorithmic efficiency has played a partial role in improving performance, hardware development (including parallelism and increased scale and spending) is the primary driver behind the progress of artificial intelligence [5, 6]. Unlike their classical counterparts, QNNs are able to learn a generalized model of a dataset from a substantially smaller training set [7–9] and typically have the potential to do so with polynomially or exponentially simpler models [10–12]. Thus, they provide a promising opportunity to subvert the scaling problem encountered in classical machine learning [1, 13–18], which presents a serious challenge for data-intensive problems that are increasingly bottle-necked by hardware limitations [19–21].

Nonetheless, even for a small dataset, training QNNs requires on the order of a million circuit evaluations. This is a consequence of the multiplicative number of data points, evaluations required for calculating the gradient [22], and iterations before a solution is reached. This makes them a relatively challenging and resource-intensive use case for quantum processing units (QPU). Therefore, QNNs require stable, on-demand, and accurate quantum circuit execution. A plethora of different options for executing quantum circuits exist. These are either physical QPUs or classical hardware simulating quantum behaviour. In both cases multiple vendor options and services are available. Establishing the combination of software and hardware that provides the optimum runtime, cost, and accuracy is crucial to the future of democratizing quantum software development.

In this study, different pairings of software development kits (SDKs) and hardware platforms are compared in order to determine the fastest and most cost-efficient route to developing novel quantum algorithms. This benchmark is performed using QNNs, which represent the most general form of a quantum algorithm. The benchmark indicates an advantage in using the QMware basiq simulator for circuits with 2 to 26 qubits, AWS SV1 for 28-34 qubits, and QMware basiq for 36-40 qubits. Additionally, QPUs from four different vendors (IonQ, Oxford Quantum Circuits (OQC), IBM, and Rigetti) were benchmarked for runtime, accuracy, and cost. The results show QPUs could become time-competitive in a practical use case for circuits with 30 qubits or more. However, the current low fidelity attained by many of these systems precludes their application to industrial problems.

This investigation is not an exhaustive benchmark of all available systems, and it is worth acknowledging the existence of other state-of-the-art services. This includes hardware quantum simulators such as IBM's simulator statevector and Atos's Quantum Learning Machine [23], as well as software backends such as the IBM Qiskit machine learning suite and the Qulacs package [24, 25]. It also includes QPUs such as IBM's Eagle processor [26], Honeywell's System Model H1 [27], and Google's Sycamore processor [28]. The authors also acknowledge other works that involved a similar technique in performing benchmarks on quantum hardware. These include hardware specific performance benchmarks such as Refs. [29, 30] as well as metric-specific tests such as Refs. [31–37].

The work is organized in five sections. Sec. II describes the methodology, including the benchmark algorithms, the hardware and software tested, as well as the results of the runtime benchmark. Sec. III details the cost of executing the proposed quantum circuits on various QPUs and compares their fidelity. Sec. IV discusses the execution of large quantum circuit, with as many as 40 qubits, and the implementation of multi-threading in simulators. Finally Sec. IV C provides a summary of the findings, and outlines key challenges for the quantum computing industry.

II. RUNTIME BENCHMARKING

Quantum simulators are bipartite systems, consisting of a software library and the hardware on which the software is run. Both play a crucial role in the development and execution of a quantum circuit. Although the various software libraries available for quantum simulation are often implemented in a hardware-agnostic way, the *internal* implementation of the linear algebraic methods and the manner in which quantum logic gates are compiled will have a significant impact on performance. This is true for the execution of any quantum circuit, but particularly relevant for gradient calculations when optimizing a QNN during the training phase, due to the use of gradient-based optimization techniques. In particular, the standard *parameter-shift* method of calculating the gradient of the circuit output with respect to each of the n trainable parameters increases the number of expectation values evaluated by a factor of $2n$. By comparison, the forward pass of a trained QNN requires evaluating just a single expectation value, which can usually be obtained with fewer than one thousand circuit shots. The specification of the simulator hardware also plays an important role in the ability to quickly and efficiently optimize a variational quantum algorithms. Moreover, the synergy between software and hardware influences how much computational overhead is required and the ease with which quantum algorithms can be designed, tested, and deployed.

For many quantum computing applications, the open-source PennyLane Python library is extremely popular [38]. It is also the recommended quantum simulation library for the AWS Braket computing service, which provides a performance-optimized version of the PennyLane library [39]. PennyLane offers a variety of qubit devices. The most commonly used is the `default.qubit`, which implements a Python backend using NumPy, TensorFlow, JAX, and PyTorch. More recently the `lightning.qubit` device was introduced, which implements a high-performance C++ backend [38]. QMware's cloud computing service provides a quantum simulator stack which supports open-source, hardware-agnostic libraries, such as PennyLane, in addition to QMware's bespoke quantum computing Python library *basiq* [40]. The basiq library also supports a PennyLane plugin, which translates circuits built using the Pennylane SDK into circuits that can be executed using the basiq backend.

A. Methodology

Hardware	Qubits	Native Gates
Rigetti Aspen M-2	80	RX RZ CZ, CP
IBMQ Falcon r5.11	27	I, CX, IFELSE, RZ, SX, X
IonQ Harmony	11	GPI, GPI2, MS
OQC Lucy	8	I, ECR, V, X, RZ

TABLE I. Quantum processing units used in the hardware benchmarking tests, their qubit counts, and native gates.

Initially the performance of the QMware HQC4020 simulator is compared to the performance of the AWS Braket ml.m5.24xlarge simulator. In both instances the PennyLane `lightning.qubit` backend is used in order to evaluate the performance of the underlying hardware systems – referred to as QPL and APL, respectively. The performance of the QMware basiq library is then benchmarked using both a native basiq implementation of the QNN and HQNN. This runtime benchmark is executed on the QMware HQC4020 simulator using 384 vCPUs across all circuit sizes. Results are compared to the high-performance AWS Braket SV1 simulator (ASV), as well as the previous QPL and APL benchmark. Finally, the runtime performance of these simulator stacks is also compared to runtimes achieved for (H)QNN inference (forward pass) using real QPUs. The QPUs included are IonQ's Harmony, OQC's Lucy, Rigetti's Aspen M-2, and IBM Quantum's Falcon r5.11.

The ml.m5.24xlarge AWS notebook instance provides 96 vCPUs and 384 GiB of random access memory (RAM). By comparison the QMware HQC4020 simulator has 12 TB of RAM and 384 vCPUs in total, a maximum of 48 vCPUs of which are utilized throughout the benchmarking tests executed on the QMware service. The exception to this is when benchmarking the PennyLane Lightning qubit, which implements no parallelization for the parameter-shift method. Hence the results for the QMware PennyLane `lightning.qubit` (QPL) and AWS

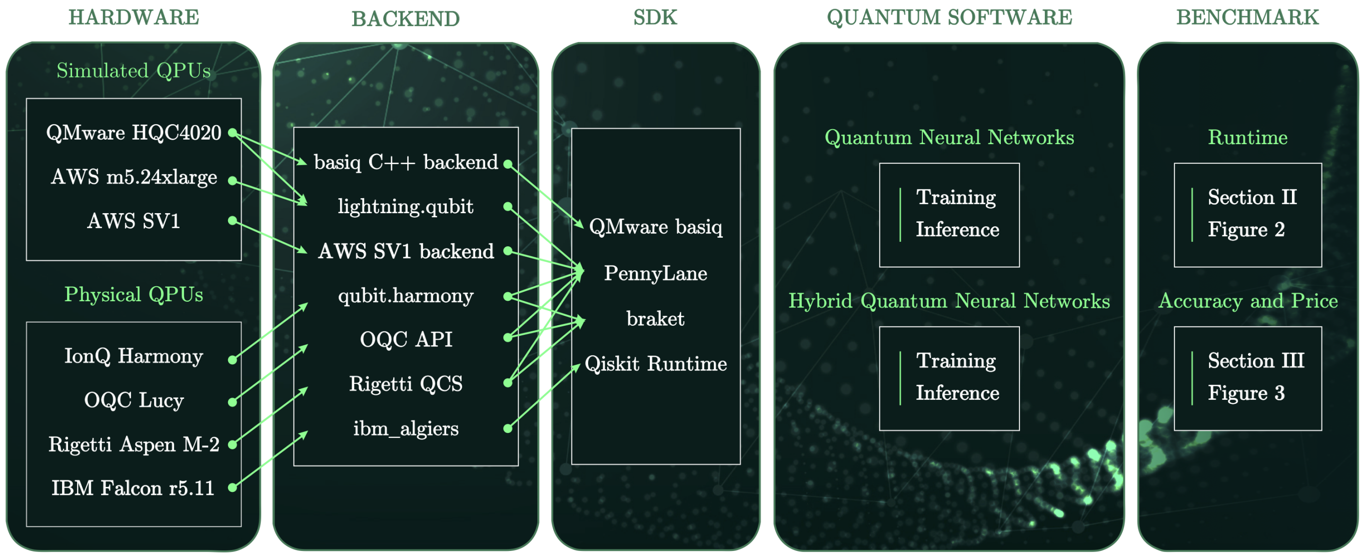

FIG. 1. The scheme for the benchmark. Simulated and native QPU stacks are benchmarked in the present work using open-source software and proprietary software on AWS, IonQ, OQC, Rigetti, IBM, and QMware.

PennyLane `lightning.qubit` (APL) benchmark represents single-core calculations on both hardware services.

The metric used in benchmarks is the training time per epoch per training sample, which is measured using the Python `time` library. The expectation value of each circuit measurement, corresponding to the output prediction of the (H)QNN, is obtained using 1000 circuit shots. Measuring the execution time for multiple circuit shots has the effect of reducing the proportion of circuit initialization time in the quoted runtime for each stack. In practice, quantum circuits usually require hundreds or thousands of circuit shots to obtain an accurate expectation value. Benchmarks are performed for a range of circuit sizes (number of qubits). This is achieved by varying the dimension of the dataset, $n_d \in [1, 15]$, which increases linearly with the number of qubits, $n_q = 2n_{\mathrm{d}}$ (see Sec. II B for details).

Although the IBM Quantum Cloud development platform allows users to retrieve quantum processor runtimes, the AWS Braket computing platform precludes the possibility of measuring low-level process times. As a result all times quoted, for both AWS and QMware benchmark, include the interaction with the runtime environment and the backend that compiles, initializes, and invokes the quantum circuit. This follows the standard practice in quantum benchmarks and reflects a real-world use case encompassing the full software and hardware stack [41]. For the benchmark involving the QPU devices available via AWS Braket, quoted times also include any potential queue time incurred on the vendor backend. The proportion of the total runtime this constitutes varies according to the number and complexity of tasks submitted by other users of the same QPU (as detailed in the Amazon Braket Developer Guide) and cannot be accurately estimated through the AWS user interface.

B. Benchmark Dataset

The dataset for the benchmarks presented here is an n-dimensional abstracted version of the well-documented two-dimensional `sklearn.datasets.make_circles` for binary classification tasks [42]. It consists of points sampled from two concentric circular shells, distributed randomly about two nominal mean radii, $r_{\mathrm{inner}} < r_{\mathrm{outer}}$. The distribution of the data points about the mean radius is described by a normal distribution, and both shells use the same standard deviation. The method for creating the dataset is adapted from that proposed by [43] for sampling points on a sphere. First, an n_d-dimensional vector is created with components v_i sampled randomly from a standard normal distribution with mean $\mu = 0$ and standard deviation $\sigma = 1$:

$$\vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{n_d} \end{bmatrix}, \quad v_i \in \mathcal{N}(0, 1). \tag{1}$$

The vector is then normalized to length r to obtain a point sampled randomly with uniform probability from the surface of an n-sphere. A vector of random noise, $\vec{\rho}$, sampled independently from the distribution $\mathcal{N}(0, \sigma^2)$, is applied to each component of the vector:

$$\vec{x} = r\frac{\vec{v}}{\|\vec{v}\|} + \vec{\rho}, \quad \rho_i \in \mathcal{N}(0, \sigma^2). \tag{2}$$

Data points $\vec{x}$ are sampled from two such distributions to create an outer shell with classification label $y_i = 0$ and inner shell with classification label $y_i = 1$. In the present work $r_{\mathrm{outer}} = 1.0$ is used to create points in the outer shell and $r_{\mathrm{inner}} = 0.3$ to create points in the inner

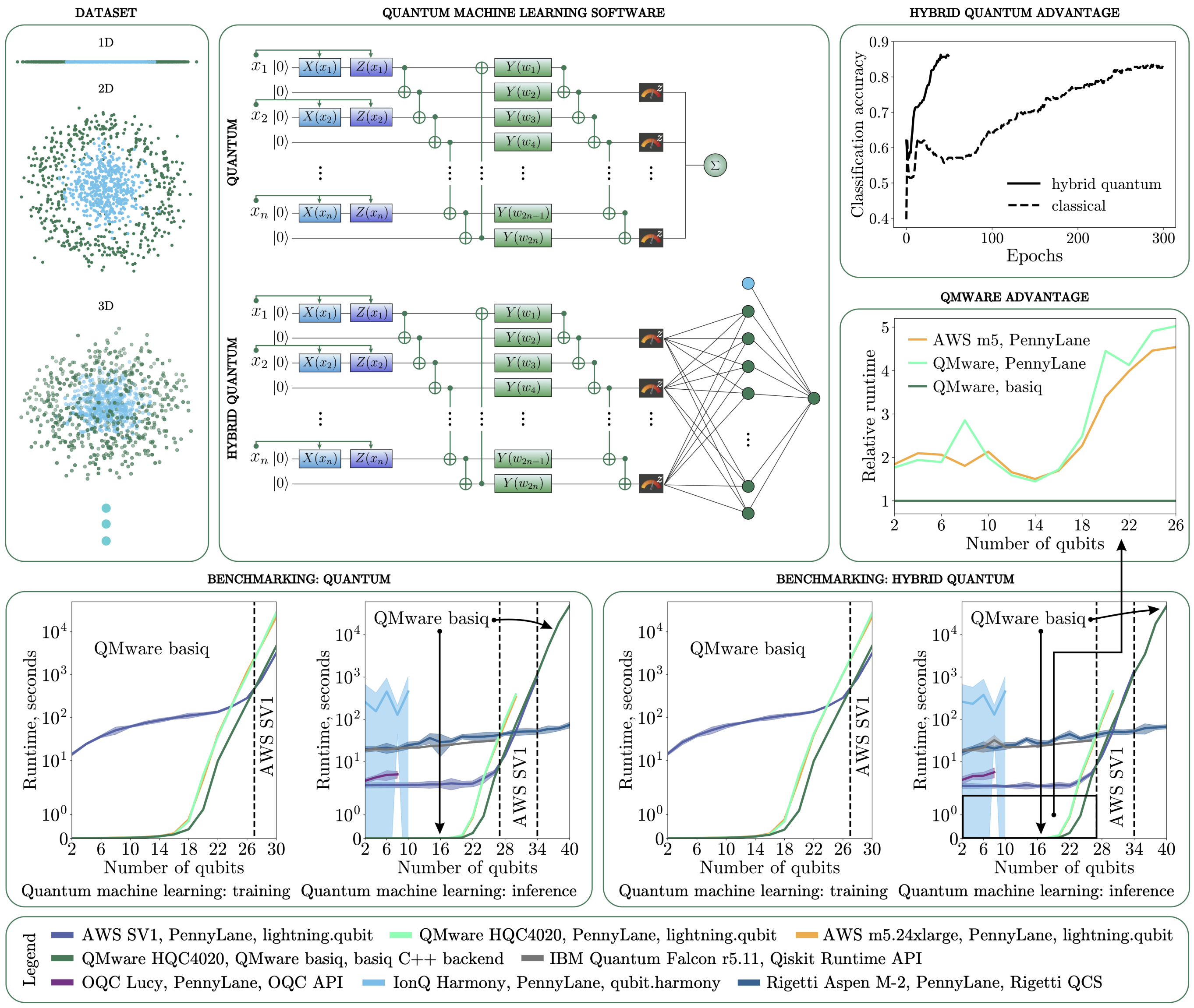

FIG. 2. Overview of benchmarking methodology and results. Boxes clockwise from top left: examples of the n_d-dimensional dataset used for benchmarking for $n_d = 1, 2, 3$, each containing $m = 500$ data points; the architecture of the pure quantum (top) and hybrid (bottom) neural networks used for the benchmarking calculations; the classification accuracy of the HQNN and an equivalent classical neural network, illustrating the improved learning rate of a hybrid approach; the relative runtime of different software and hardware stacks for a range of circuit sizes, showing the performance advantage of the basiq SDK using QMware's cloud computing service; training (left) and inference (right) runtime for the HQNN on various simulator software and hardware stacks; training (left) and inference (right) runtime for the pure QNN on various simulator software and hardware stacks (see legend at bottom). Dashed lines in the runtime plots illustrate the most performant stack for a given range of qubits. The inference plots in both cases also illustrate the runtime using real QPUs.

shell, with $\sigma = 0.2$ for both shells. The dimension of the dataset determines the number of features used as input to the neural network. By linearly increasing the dimension of the dataset, circuits with a varying number of qubits ($n_q = 2n_d$) can be benchmarked without changing the underlying rubric of the classification problem itself. Examples of the one-, two- and, three-dimensional datasets are shown in Fig. 2.

C. Learning Models

The following sections describe the architecture of the hybrid quantum-classical (HQNN) and pure QNNs used in the benchmarking tests. In all cases, the networks are trained using a binary cross-entropy loss function and the Adam optimizer with a learning rate of $\alpha = 0.3$ [44].

1. Quantum Neural Network

The QNN used in this benchmark consists of a multi-qubit variational quantum circuit, and is based on the model proposed by [7] The model employs a sequential R_x R_z two-axis rotation encoding scheme to embed each of the data features in a single qubit state on the odd-numbered qubits. The feature encoding is followed by an entangling layer of sequential 'nearest-neighbor' CNOT gates across all qubits. A layer of trainable single-axis R_y rotation are then applied to each qubit, followed by a final layer of sequential 'cascading' CNOT gates across all qubits. Finally, the expectation value of the Pauli-Z operator (σ_z) is measured for each of the even-numbered qubits. The mean expectation value across the n_d measurement qubits is interpreted as a probability of the input belonging to the class $y_i = 1$.

To obtain the gradients of the loss function with respect to each of the parameters, the standard analytical *parameter-shift* algorithm is used, in which the gradient of an expectation value, with respect to the i-th parameter, is obtained via measurement of two additional expectation values. This method requires a total of $2n_w+1$ circuit evaluations to obtain the gradient of the loss with respect to n_w circuit parameters and thus scales linearly with the number of trainable parameters. Unlike the more efficient *adjoint* and *backpropagation* methods, which are usually available for quantum simulators, parameter-shift is the only currently available algorithm that can be implemented natively on a QPU. It therefore provides an upper bound on the cost of training a QNN using a QPU, and on the runtime of the benchmarked simulators [45, 46].

2. Hybrid Quantum Neural Network

The HQNN is a bipartite model consisting of a QNN and a multi-layer perceptron (MLP) classical neural network, with the expectation value of each measurement qubit in the QNN used as an input feature for the first layer of the MLP. The quantum component of the HQNN follows the same architecture as described in Sec. II C 1, and uses the same parameter-shift method to obtain the gradients of the expectation values. The MLP is built using the PyTorch library and contains three linear layers with sizes $[n_d, 40, 1]$, with ReLU and Sigmoid activation functions applied to the input and hidden layer, respectively [47]. The gradients in the classical component of the HQNN are computed using the standard back-propagation algorithm, via the native implementation in the PyTorch library. For the inference benchmark that utilizes QPUs or the AWS SV1 device, the classical part of the circuit is executed using the AWS ml.m5.24xlarge compute instance in all cases except for IBM Quantum Falcon r5.11 whose classical part was executed on QMware HQC4020.

D. Results

This section presents the results of the benchmark methodology outlined in Sec. II A. The results of the benchmark are illustrated in Fig 2. A table of these results is also given in App. IV C, Tab. III and Tab. IV. The measured runtime values (runtime per epoch, per training sample) are averaged across 100 repeats in order to obtain a mean and standard deviation for circuits up to 24 qubits in size for the training benchmark, and 26 qubits for the inference benchmark. A similar approach to averaging is impractical for larger circuit sizes, owing to the significantly longer total runtime. Thus, only a single value with no standard deviation is quoted for circuits larger than 24 qubits. In all cases Chauvenet's criterion is applied in order to filter anomalous runtime measurements that arise due to extraneous hardware processes [48].

Label	Hardware	SDK	Backend
QBN	QMware HQC4020	basiq	basiq C++
QPL	QMware HQC4020	PennyLane	lightning.qubit
APL	AWS ml.m5.24xlarge	PennyLane	lightning.qubit
ASV	AWS SV1	PennyLane	SV1

TABLE II. The abbreviations associated with each hardware, SDK, and backend.

1. Training

In general, the QMware HQC4020 and the AWS ml.m5.24xlarge hardware achieve similar performance using the Pennylane Lightning backend (QPL and APL, respectively - see Tab. II for a list of abbreviations). When benchmarking with the QNN, QPL performs similarly to APL for circuits with less than 27 qubits in size, with a relative runtime of $t_{\mathrm{QPL}}/t_{\mathrm{APL}} = 0.90(17)$. When benchmarking with the HQNN the average relative runtime is $t_{\mathrm{QPL}}/t_{\mathrm{APL}} = 0.95(16)$.

Comparing the QMware HQC4020 hardware using a native basiq implementation (QBN) to the Pennylane Lightning implementation (QPL) shows a clear advantage for the native basiq implementation across all circuit sizes for both QNNs and HQNNs. The QBN implementation achieves an average speedup of $t_{\mathrm{QBN}}/t_{\mathrm{QPL}} = 0.56(32)$ over QPL in the QNN benchmark, and $t_{\mathrm{QBN}}/t_{\mathrm{QPL}} = 0.54(30)$ for HQNNs. It is also notable that the native QMware implementation is most performant for models using fewer than more than 20 qubits, where an average speedup of $t_{\mathrm{QBN}}/t_{\mathrm{QPL}} = 0.22(4)$ is obtained for the QNNs and $t_{\mathrm{QBN}}/t_{\mathrm{QPL}} = 0.21(2)$ for the HQNNs.

The AWS Braket SV1 device is a high-performance managed device designed for simulating large quantum circuits up to a maximum of 34 qubits. Correspondingly, it outperforms QBN for circuits with 28 or 30 qubits with an average relative runtime across QNNs and HQNNs of

$t_{\mathrm{QBN}}/t_{\mathrm{ASV}} = 1.35(11)$. Conversely, it has very poor performance for small and medium circuits, with a relative runtime approximately two to four orders of magnitude slower than the other available simulator stacks for circuits with fewer than 20 qubits. Across all circuit sizes smaller than 28 qubits, the QBN implementation outperforms ASV for both QNNs and HQNNs. The average relative runtime is $t_{\mathrm{QBN}}/t_{\mathrm{ASV}} = 0.000\,41(25)$ for circuits with 14 qubits or less, $t_{\mathrm{QBN}}/t_{\mathrm{ASV}} = 0.0049(35)$ for circuits with 16 to 20 qubits, and $t_{\mathrm{QBN}}/t_{\mathrm{ASV}} = 0.36(29)$ for circuits with 22 to 26 qubits.

2. *Inference*

When training a QNN the number of circuit evaluations increases linearly with the number of trainable parameters. In contrast, the inference (forward pass) requires only a single evaluation of the output expectation values. As a result, the total runtime is reduced significantly. The results of the inference runtime benchmark broadly follow the same trends as for the training runtime. One exception to this are the QPL and APL relative runtimes, which are identical within one standard deviation for circuits less than 16 qubits in size. The average relative runtime is $t_{\mathrm{QPL}}/t_{\mathrm{APL}} = 1.04(10)$ for QNNs and $t_{\mathrm{QPL}}/t_{\mathrm{APL}} = 1.03(22)$ for HQNNs. For larger circuits, APL performs marginally better than QPL with $t_{\mathrm{QPL}}/t_{\mathrm{APL}} = 1.11(10)$ and $t_{\mathrm{QPL}}/t_{\mathrm{APL}} = 1.12(9)$ for QNNs and HQNNs, respectively.

Crucially, the inference tests include a benchmark of physical QPUs, as listed in Tab. I. For low numbers of qubits the QPU runtimes are orders of magnitude longer than their simulator counterparts. This is likely due to the additional overhead incurred in compiling and initializing a QPU results, as well as the queue time incurred on the vendor side, where multiple AWS users may be accessing the QPU device simultaneously. On the other hand, the runtime for QPUs increases linearly with the number of qubits. For large numbers of qubits, the exponentially increasing simulator runtime exceeds the fixed time cost associated with QPUs. This results in a 'threshold' circuit size above which it becomes exponentially faster to execute a QNN using a QPU device.

The limited number of qubits available for many QPUs means that, in most cases, this threshold is not attainable. Of the three QPUs in the present study, only Rigetti's Aspen M-2 has a sufficient number of qubits to be time-competitive with the simulator stacks tested. The threshold occurs at 30 qubits for QBN and ASV where the relative runtime is $t_{\mathrm{ASV}}/t_{\mathrm{M\text{-}2}} = 1.03$, $t_{\mathrm{QBN}}/t_{\mathrm{M\text{-}2}} = 1.53$ for the QNN and $t_{\mathrm{ASV}}/t_{\mathrm{M\text{-}2}} = 1.02$, $t_{\mathrm{QBN}}/t_{\mathrm{M\text{-}2}} = 1.51$ for the HQNN. For smaller circuits sizes OQC's Lucy produces runtimes that are faster by a factor of $t_{\mathrm{Lucy}}/t_{\mathrm{M\text{-}2}} = 0.22(3)$ compared to Rigetti's Aspen M-2. In contrast, the runtime for IonQ's Harmony QPU is on average a factor $t_{\mathrm{Harmony}}/t_{\mathrm{M\text{-}2}} = 13.5(52)$ slower than Aspen M-2. The IBMQ Falcon r5.11 QPU performs similarly to the Rigetti Aspen M-2, with an average relative runtime of $t_{\mathrm{ASV}}/t_{\mathrm{M\text{-}2}} = 0.87(18)$.

Notably, vendor management of the QPUs results in a runtime that varies significantly from job to job. The percentage variance in runtime measured for inference on a QPU is generally higher than for simulator execution. In particular for IonQ's Harmony device, the average percentage standard deviation across the 100 repeat measurements is on the order of %125, and %16, %21, %7 for Aspen M-2, Lucy, and Falcon r5.11, respectively.

III. ACCURACY AND COST ANALYSIS

A. Cost Evaluation

As described in Sec. I a primary consideration in developing and testing QNNs and HQNNs is the financial cost of training a network. In general, the pricing structure of publicly available QPUs is such that the training cost is proportional to the number of distinct quantum circuits that must be evaluated during the training process. When using the parameter-shift method for gradient calculations, the number of distinct quantum circuits is proportional to the number of trainable parameters. Hence, for the QNN and HQNN considered in Sec. II C 1 and Sec. II C 2, the training costs scales linearly with the number of qubits. For an increasing number of epochs, and for an increasing number of training samples, this rapidly results in millions or billions of distinct quantum circuits that must be initialized and evaluated on the QPU.

In the present work, QPUs are accessed through AWS Braket and IBM Cloud Qiskit Runtime. The AWS pricing scheme includes both a per-task cost, incurred for the execution of a given quantum circuit, and a per-shot cost which is applied to the number of shots specified for that quantum circuit. By comparison, Qiskit Runtime implements a pricing scheme on the basis of runtime with a fixed price per second for executing a quantum circuit. Fig. 3(b) illustrates an example of how the estimated cost scales with circuit size for the four QPUs presented in this work. The consequence of this QPU pricing structure is that for circuits larger than a few bits, training a QNN on a QPU becomes prohibitively expensive. Prices range from approximately 1000 USD for a two-qubit QNN using Rigetti's Aspen M-2 or OQC's Lucy, to more than 10×10^6 USD for a 26 qubit QNN using IBM's Falcon r5.11.

In contrast, the forward pass of a QNN does not entail a gradient calculation and thus requires only a single task with around 100-1000 circuit shots. The cost of using a QPU for the inference stage of a trained QNN is considerably less. For the QPUs accessed through AWS Braket, the cost is fixed for a given number of shots, irrespective of circuit size. For QPUs accessed through Qiskit Runtime, the cost increases linearly with circuit size (QPU runtime increases linearly with circuit size).

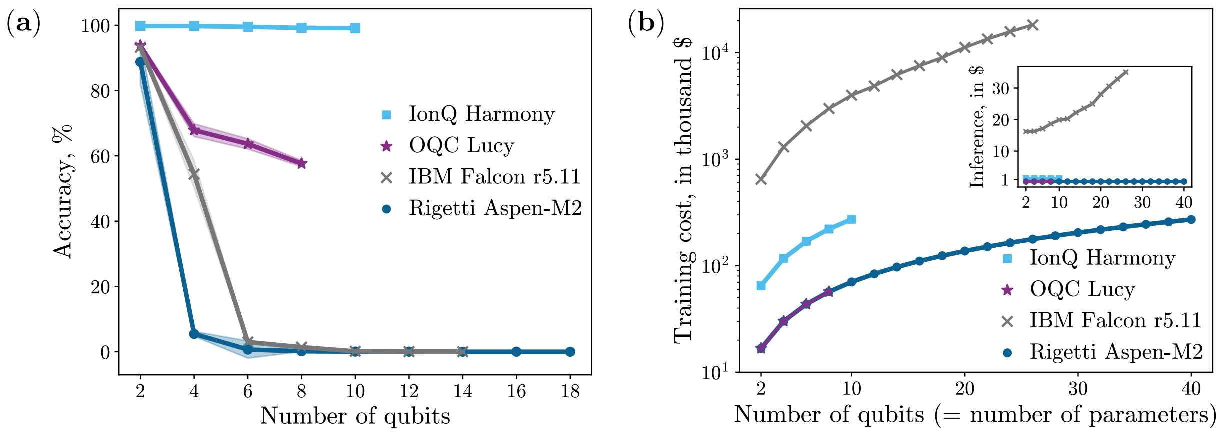

FIG. 3. Comparison of circuit fidelity (left), training cost (right) and inference cost (inset) for various publicly available QPUs. The training price estimates are calculated on the assumption that the model is trained over 100 epochs, with 100 training samples and 1000 circuit shots per expectation value. For IBM Falcon r5.11 the prices are obtained in CHF and the conversion is calculated using a rate of 1.06 USD = 1.00 CHF.

A typical forward-pass of a QNN can be executed at a cost of approximately 1 USD for QPUs accessed through AWS Braket, or between 10 - 40 USD for QPUs accessed through Qiskit Runtime.

B. Accuracy Evaluation

Understanding the role noise plays in executing quantum circuits is critical to leveraging quantum computers. Vendors often provide esoteric measures of fidelity, gate accuracy and characteristic timescales. Although these are valuable metrics for assessing the relative performance of different QPUs, evaluating the overall effect of these different sources of error on a typical quantum circuit can be challenging. A holistic measure of noise in a quantum circuit can be achieved with a straightforward empirical procedure.

First, the parameters of all trainable gates are fixed at a value of $\pi/4$. The input feature values are each set to $\pi^2/4$, and the QNN is augmented by applying the adjoint of the entire circuit prior to measurement. In a noiseless circuit this results in a final state vector with zero amplitude for all basis states except the computational ground state, i.e $|00...0\rangle$. In a physical QPU, various noise sources degrade the fidelity of the circuit, resulting in a non-zero probability of measuring one of the other $2^{n_q}-1$ computational basis states. An accuracy measure is then obtained by counting the proportion of states measured in the computational ground state over 1000 circuit shots. This is commonly referred to as the fidelity of a quantum state, $F = |\langle 00...0|\psi\rangle|^2$. In this case the fidelity of the final quantum state is measured relative to the computational ground state. This fidelity measurement is repeated by executing 10 such jobs on each QPU to obtain a mean accuracy. The state vectors needed to obtain fidelities for individual shots are not available through the PennyLane SDK with AWS. Consequently, the AWS braket software library is used instead to construct the same quantum circuits described in Sec. II C for physical QPU accuracy tests presented here, with the exception of the IBM Falcon r5.11 which was tested on QMware and accessed through IBM Cloud's Qiskit Runtime API.

The results, shown in Fig. 3(a) and Tab. IX, vary significantly between QPUs. The IonQ Harmony device attains greater than 99% fidelity across circuits sizes from two to ten qubits. The fidelity of the OQC Lucy device is high for small circuits with only two qubits, but degrades significantly for larger circuits, with 67 ± 2 %, 64 ± 2 % and 57.6 ± 0.9 % fidelity for four, six, and eight qubits, respectively. The performance of the Rigetti Aspen M-2 and IBMQ Falcon r5.11 QPUs is similarly high for the two qubit circuit, with fidelities of 89 ± 6 % and 93.2 ± 0.4 %, respectively. However the fidelity of these devices approaches zero for circuits larger than approximately eight qubits.

IV. DISCUSSION AND SUMMARY

A. Large Quantum Circuits

The amount of random access memory (RAM) utilized in simulating any noiseless quantum circuit is a function of the dimension of the vector space, and hence grows exponentially with the number of qubits. An n-qubit state is specified by 2^n complex amplitudes, and thus requires approximately 16×2^n bytes of memory. This equates to approximately 16 GB of RAM for a 30 qubit circuit.

QNNs using large numbers of qubits are highly susceptible to the barren plateau phenomenon, where the gradients of a QNN with randomly initialized parameters van-

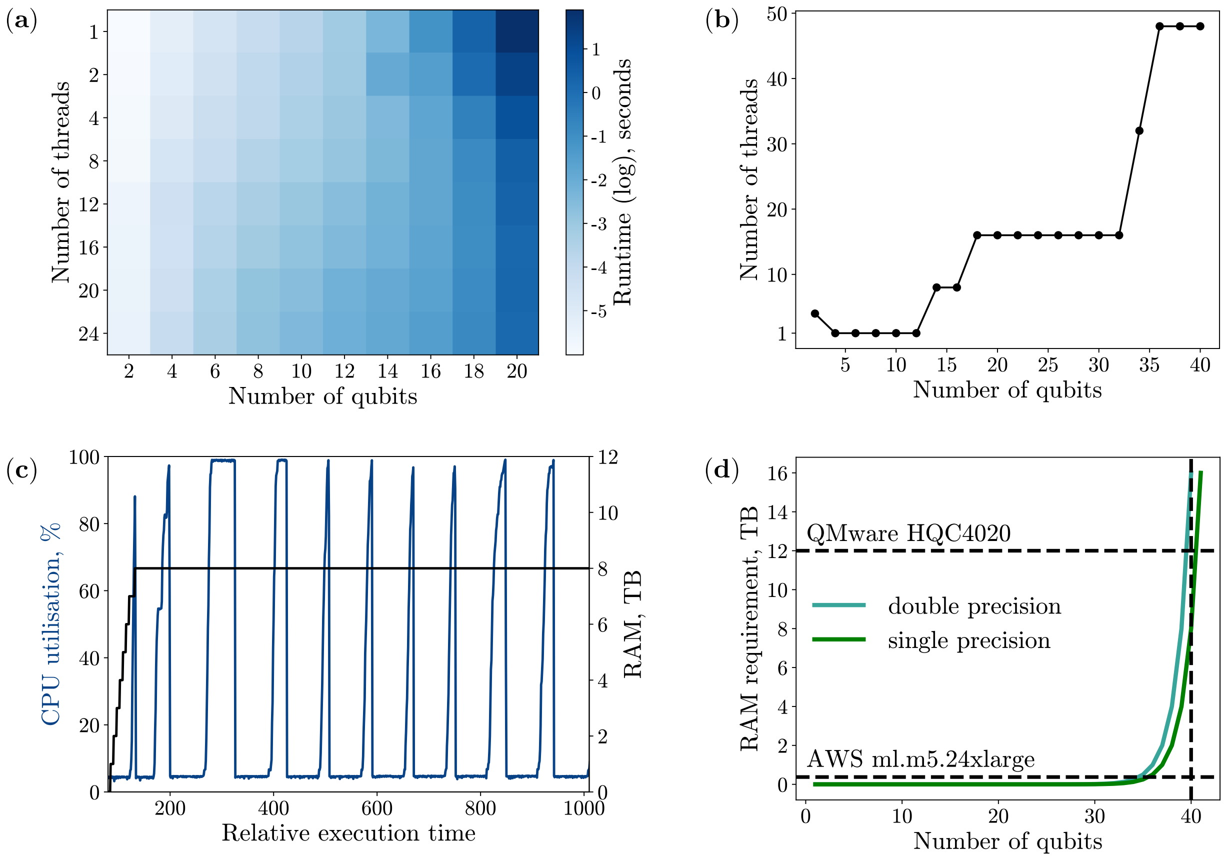

FIG. 4. Hardware statistics for the QMware HQC4020 simulator. (a) The optimal number of threads for simulating an n-qubit quantum circuit. Faster runtimes are indicated with a paler shade, demonstrating how the optimal number of threads varies for increasing circuit size. (b) a plot of the optimum thread count against number of qubits (c) a plot of total CPU and RAM utilization over time during the execution of a 40 qubit QNN. (d) memory required to store the state vector for an increasing number of qubits.

ish exponentially as the number of qubits increases [49]. This could present a serious obstacle to training a QML model with a large number of qubits unless a variety of mitigation methods are employed [50–52]. Solving the barren plateau problem is crucial to achieving highly expressive models that could outperform classical machine learning methods on complex datasets. Access to development services that are able to simulate QNNs with a large number of qubits is essential to solving the barren plateau problem.

To explore the performance of QMware's HQC4020 with an increasing number of qubits, additional benchmarks are performed for QNNs and HQNNs with up to 40 qubits. A 40 qubit simulation is achievable by reducing the floating point precision to single precision (32 bit) representation, for all other circuit sizes a double precision (64 bit) is retained. Fig. 4(d) illustrates the exponential increase in memory usage for a range of circuit sizes up to 40 qubits, for both single and double precision representations.

Tab. XI gives the inference runtimes for circuits with up to 40 qubits, up to a maximum of 34 qubits in the case of ASV. The runtimes for the QBN and ASV simulators increase exponentially, with QBN reaching a runtime greater than 13 hours for the 40 qubit QNN. Owing to the long simulator runtimes for large circuits, multiple repeats are not possible, and thus it is hard to draw meaningful conclusions from the single QBN and ASV trials presented for circuits larger than 27 qubits. Nonetheless, in the case of QNNs, ASV achieves a relative runtime of $t_{\mathrm{ASV}}/t_{\mathrm{QBN}} = 1.34(9)$ for circuits with 28 to 34 qubits. In the case of HQNNs there is no clear advantage, with $t_{\mathrm{ASV}}/t_{\mathrm{QBN}} = 0.95(22)$.

B. Multi-threading

The runtime performance for circuits with many qubits can be improved using various parallelization techniques. A common method for achieving a substantial increase in runtime performance for linear algebra operations is to compute matrix-vector and matrix-matrix products with the aid of multi-threading. The QMware basiq library provides native support for a multi-threaded approach to low-level C++ linear algebra operations. However, determining the optimal number of threads to execute a circuit with a given number of qubits is not straightforward. Fig. 4(a) illustrates a cross-sectional study of runtime for circuits with up to 20 qubits with multi-threading across as many as 24 threads. In Fig. 4(b) the optimum number of threads for a given qubit count is shown. These results can be applied generally to achieve best-in-class performance with the QMware HQC2040 using the basiq software library.

PennyLane provides some parallelization for gradient calculations using the adjoint operator method [46], which is applicable to parameterized quantum algorithms. However, the authors are not aware of any low-level parallelization in the PennyLane library for general linear algebra calculations encountered in generic quantum circuits.

C. Conclusion

This work presents a comprehensive study of various quantum computing platforms, using both simulated and physical quantum processing units. The results presented in Sec. II D demonstrate a clear runtime advantage for the QMware basiq library executed on the QMware cloud computing service. Relative to the next fastest classical simulator, QMware achieves a runtime reduction of up to 78% across all algorithms with fewer than 27 qubits. In particular, the QMware basiq library achieves a runtime reduction of 0.56(32) relative to the PennyLane library for QNNs and 0.54(30) for HQNNs.

The QMware HQC4020 hardware benchmarked against AWS ml.m5.24xlarge achieves a comparable relative runtime of 0.90(17) for QNNs and 0.95(16) for HQNNs. Thus, the advantage offered by the QMware cloud computing service is primarily due an harmonious interplay between software and hardware. This performance advantage can be attributed to the superior multi-threading support present in the basiq library. AWS also provides the SV1 simulator, which performs marginally better than QMware basiq for large circuits with more than 27 qubits in the case of QNNs (up to the SV1 maximum of 34 qubits). There is no clear advantage for either QMware or SV1 in the case of HQNNs. Additionally, QMware is the only tested simulator that has the capability of simulating circuits with 34-40 qubits.

The price and scarcity of quantum hardware means it is more time and cost efficient to develop algorithms and train QNNs with quantum simulators such as Amazon Web Services Braket or QMware's cloud computing service. This is particularly true for variational quantum algorithms and QNNs which represent a promising utilization of quantum technologies in artificial intelligence on NISQ computers. In contrast to the exponential runtime scaling encountered in quantum simulators, the runtime of QPUs scales linearly with the circuit size. Publicly available QPUs are already able to achieve runtime improvements over simulator hardware for large numbers of qubits. For example, Rigetti's Aspen M-2 is able to execute 40 qubit circuits in approximately one minute, which is less than 0.02% of the runtime measured for QMware's simulator.

As quantum hardware improves and the number of qubits available grows, it will become possible to gain a substantial runtime advantage over simulator hardware when executing large quantum circuits. However, the fidelity tests presented in Sec. III B indicate that accurate inference with such a large quantum circuit is not yet possible. Moreover, the cost of accessing these quantum devices makes training a QNN on currently available QPUs prohibitively expensive. Owing to the exponential computational space, QNNs with relatively few qubits are able to tackle challenging data science and industry problems. Thus, the key to success in the field of quantum computing is to improve the cost and the accuracy of QPUs, and integrating them well within classical infrastructure. The hybrid interplay between quantum and classical machines is the key to seamlessly harness the best performance of QPUs and simulators depending on the use case.

[1] Javier Alcazar, Vicente Leyton-Ortega, and Alejandro Perdomo-Ortiz. Classical versus quantum models in machine learning: Insights from a finance application. Machine Learning: Science and Technology, 1(3):035003, 2020.

[2] Asel Sagingalieva, Andrii Kurkin, Artem Melnikov, Daniil Kuhmistrov, et al. Hyperparameter optimization of hybrid quantum neural networks for car classification. arXiv preprint arXiv:2205.04878, 2022.

[3] Manuel S Rudolph, Ntwali Bashige Toussaint, Amara Katabarwa, Sonika Johri, Borja Peropadre, and Alejandro Perdomo-Ortiz. Generation of high-resolution handwritten digits with an ion-trap quantum computer. Physical Review X, 12(3):031010, 2022.

[4] Asel Sagingalieva, Mohammad Kordzanganeh, Nurbolat Kenbayev, Daria Kosichkina, Tatiana Tomashuk, and Alexey Melnikov. Hybrid quantum neural network for drug response prediction. arXiv preprint arXiv:2211.05777, 2022.

[5] Liane Bernstein, Alexander Sludds, Ryan Hamerly, Vivienne Sze, Joel Emer, and Dirk Englund. Freely scalable and reconfigurable optical hardware for deep learning. Scientific Reports, 11(1):3144, 2021.

[6] Danny Hernandez and Tom B. Brown. Measuring the Algorithmic Efficiency of Neural Networks. arXiv preprint arXiv:2005.04305, 2020.

[7] Michael Perelshtein, Asel Sagingalieva, Karan Pinto, Vishal Shete, et al. Practical application-specific advantage through hybrid quantum computing. arXiv preprint arXiv:2205.04858, 2022.

[8] Matthias C. Caro, Hsin-Yuan Huang, M. Cerezo, Kunal Sharma, Andrew Sornborger, Lukasz Cincio,

and Patrick J. Coles. Generalization in quantum machine learning from few training data. Nature Communications, 13(1):4919, 2022.

[9] Amira Abbas, David Sutter, Christa Zoufal, Aurelien Lucchi, Alessio Figalli, and Stefan Woerner. The power of quantum neural networks. Nature Computational Science, 1(6):403–409, 2021.

[10] Sergio Boixo, Sergei V. Isakov, Vadim N. Smelyanskiy, Ryan Babbush, Nan Ding, Zhang Jiang, Michael J. Bremner, John M. Martinis, and Hartmut Neven. Characterizing quantum supremacy in near-term devices. Nature Physics, 14(6):595–600, 2018.

[11] Rocco A. Servedio and Steven J. Gortler. Equivalences and Separations Between Quantum and Classical Learnability. SIAM Journal on Computing, 33(5):1067–1092, 2004.

[12] Diego Ristè, Marcus P. da Silva, Colm A. Ryan, Andrew W. Cross, Antonio D. Córcoles, John A. Smolin, Jay M. Gambetta, Jerry M. Chow, and Blake R. Johnson. Demonstration of quantum advantage in machine learning. npj Quantum Information, 3(1):16, 2017.

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

[14] Maria Schuld and Francesco Petruccione. Learning with Quantum Models, pages 247–272. Springer International Publishing, Cham, 2018.

[15] Dimitrios Emmanoulopoulos and Sofija Dimoska. Quantum Machine Learning in Finance: Time Series Forecasting. arXiv preprint arXiv:2202.00599, 2022.

[16] Brian Coyle, Maxwell Henderson, Justin Chan Jin Le, Niraj Kumar, Marco Paini, and Elham Kashefi. Quantum versus classical generative modelling in finance. Quantum Science and Technology, 6(2):024013, 2021.

[17] Bob Coecke, Giovanni de Felice, Konstantinos Meichanetzidis, and Alexis Toumi. Foundations for Near-Term Quantum Natural Language Processing. arXiv preprint arXiv:2012.03755, 2020.

[18] Konstantinos Meichanetzidis, Alexis Toumi, Giovanni de Felice, and Bob Coecke. Grammar-Aware Question-Answering on Quantum Computers. arXiv preprint arXiv:2012.03756, 2020.

[19] Xiaowei Xu, Yukun Ding, Sharon Xiaobo Hu, Michael Niemier, Jason Cong, Yu Hu, and Yiyu Shi. Scaling for edge inference of deep neural networks. Nature Electronics, 1(4):216–222, 2018.

[20] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel Emer. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. arXiv preprint arXiv:1703.09039, 2017.

[21] Mark Horowitz. 1.1 Computing's energy problem (and what we can do about it). In IEEE International Solid-State Circuits Conference, pages 10–14, 2014.

[22] Maria Schuld, Ville Bergholm, Christian Gogolin, Josh Izaac, and Nathan Killoran. Evaluating analytic gradients on quantum hardware. Physical Review A, 99(3):032331, 2019.

[23] Atos. Quantum Learning Machine. https://atos.net/en/solutions/quantum-learning-machine, 2022.

[24] Gadi Aleksandrowicz et al. Qiskit: An open-source framework for quantum computing. IBM Quantum, 2019.

[25] Koki Aoyama. Qulacs. https://github.com/qulacs/qulacs, 2022.

[26] Jerry Chow, Oliver Dial, and Jay Gambetta. IBM quantum breaks the 100-qubit processor barrier. https://research.ibm.com/blog/127-qubit-quantum-processor-eagle, 2021.

[27] J. M. Pino, J. M. Dreiling, C. Figgatt, et al. Demonstration of the trapped-ion quantum CCD computer architecture. Nature, 592:209–213, 2021.

[28] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, et al. Quantum supremacy using a programmable superconducting processor. Nature, 574:505–510, 2019.

[29] K. Wright, K. M. Beck, S. Debnath, J. M. Amini, et al. Benchmarking an 11-qubit quantum computer. Nature Communications, 10, 11 2019.

[30] Alexander J. McCaskey, Zachary P. Parks, Jacek Jakowski, Shirley V. Moore, et al. Quantum chemistry as a benchmark for near-term quantum computers. npj Quantum Information, 5, 2019.

[31] Salonik Resch and Ulya R. Karpuzcu. Benchmarking quantum computers and the impact of quantum noise, 2019.

[32] Thomas Lubinski, Sonika Johri, Paul Varosy, Jeremiah Coleman, et al. Application-oriented performance benchmarks for quantum computing. https://github.com/SRI-International/QC-App-Oriented-Benchmarks, 2021.

[33] Pacific Northwest National Laboratory. QASMBench benchmark suite. https://github.com/pnnl/QASMBench, 2022.

[34] Daniel Mills, Seyon Sivarajah, Travis L. Scholten, and Ross Duncan. Application-motivated, holistic benchmarking of a full quantum computing stack. Quantum, 5:415, 2021.

[35] Q-score. https://github.com/myQLM/qscore, 06 2022.

[36] Pierre-Luc Dallaire-Demers, Michal Stechly, Jerome F. Gonthier, Ntwali Toussaint Bashige, et al. An application benchmark for fermionic quantum simulations, 2020.

[37] Koen Mesman, Zaid Al-Ars, and Matthias Möller. Qpack: Quantum approximate optimization algorithms as universal benchmark for quantum computers, 2021.

[38] Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, et al. PennyLane: Automatic differentiation of hybrid quantum-classical computations. arXiv preprint arXiv:1811.04968, 2022.

[39] Amazon Web Services. Amazon braket. https://aws.amazon.com/, 2020.

[40] QMware. QMware — The first global quantum cloud. https://qm-ware.com/, 2022.

[41] Andrew Wack, Hanhee Paik, Ali Javadi-Abhari, Petar Jurcevic, Ismael Faro, Jay M. Gambetta, and Blake R. Johnson. Quality, Speed, and Scale: Three key attributes to measure the performance of near-term quantum computers. arXiv preprint arXiv:2110.14108, 2021.

[42] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, et al. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12:2825–2830, 2011.

[43] George Marsaglia. Choosing a Point from the Surface of a Sphere. The Annals of Mathematical Statistics, 43(2):645–646, 1972.

[44] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. arXiv preprint arXiv:1412.6980, 2017.

[45] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating

errors. Nature, 323(6088):533–536, 1986.

[46] Tyson Jones and Julien Gacon. Efficient calculation of gradients in classical simulations of variational quantum algorithms. arXiv preprint arXiv:2009.02823, 2020.

[47] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, et al. PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems 32, pages 8024–8035. Curran Associates, Inc., 2019.

[48] William Chauvenet. A Manual of Spherical and Practical Astronomy: Spherical astronomy. J. B. Lippincott & Company, 1863.

[49] Jarrod R. McClean, Sergio Boixo, Vadim N. Smelyanskiy, Ryan Babbush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. Nature Communications, 9(1):4812, 2018.

[50] Chen Zhao and Xiao-Shan Gao. Analyzing the barren plateau phenomenon in training quantum neural networks with the ZX-calculus. Quantum, 5:466, 2021.

[51] Tyler Volkoff and Patrick J. Coles. Large gradients via correlation in random parameterized quantum circuits. Quantum Science and Technology, 6(2):025008, 2021.

[52] Edward Grant, Leonard Wossnig, Mateusz Ostaszewski, and Marcello Benedetti. An initialization strategy for addressing barren plateaus in parametrized quantum circuits. Quantum, 3:214, 2019.

APPENDIX: RAW NUMERICAL RESULTS

In this section, the raw results of the benchmarks are provided in tabular view. Tab. III demonstrates the data obtained during the training process, whereas Tab. IV shows the data obtained during the inference process. The blank spaces represent two types of non-applicability: 1) the standard deviations of training (inference) runtimes were calculated up to 24 (26) qubits to avoid long execution times as well as limiting our carbon emission, and 2) as not all QPUs included enough qubits to fulfill all parts, their numbers are presented up to their highest even-qubit availability.

Tabs. V, VI, VII, VIII showcase the dependence of the QBN runtimes for quantum training, quantum inference, hybrid inference, and hybrid training, respectively. In all cases, it is evident that for an increasing number of threads we see a general improvement in runtimes. However, in some cases even for low-qubit circuits it is beneficial to use more than a single thread.

Furthermore, Tab. IX provides the numerical results for the QPU accuracy tests, whereas the times obtained are shown in Tab. X. These runtimes are largely in agreement with those in Tab. III. However, it is worth noting that these circuits are twice as deep as the ones used in runtime benchmarks as explained in Sec. III B.

Finally, Tab. XI shows the runtimes associated with running large quantum circuits. A characteristic feature of this limit is the scaling of the quantum hardware that is unavailable to any classical machine. This hints at a decisive quantum advantage in this region with the condition that the accuracy would also become competitive.

Quantum Neural Network																
n_{qubits}		**2**	**4**	**6**	**8**	**10**	**12**	**14**	**16**	**18**	**20**	**22**	**24**	**26**	**28**	**30**
QBN	MEAN	0.0024	0.0054	0.0099	0.0160	0.0248	0.0437	0.0866	0.1636	0.3786	1.2116	9.8348	46.9552	217.3697	1005.2324	4640.2727
	STD	0.0003	0.0002	0.0005	0.0006	0.0007	0.0163	0.0026	0.0122	0.0251	0.0283	0.0858	0.4396			
QPL	MEAN	0.0039	0.0095	0.0132	0.0216	0.0317	0.0455	0.0709	0.1724	0.7224	4.6597	36.8044	211.0829	998.0364	5139.8335	28121.3965
	STD	0.0006	0.0173	0.0002	0.0109	0.0144	0.0172	0.0184	0.0286	0.0383	0.1668	0.9413	8.1045			
APL	MEAN	0.0058	0.0099	0.0175	0.0285	0.0407	0.0587	0.0882	0.2106	0.7963	3.7189	39.8270	212.9200	1165.6600	4968.9300	22281.9225
	STD	0.0109	0.0002	0.0003	0.0094	0.0072	0.0110	0.0105	0.0179	0.0252	0.1924	0.8422	2.0854			
ASV	MEAN	14.0362	24.6182	36.7133	50.7568	62.1815	75.1483	86.7919	99.1183	112.1725	121.7680	136.4578	185.8316	285.5654	794.7542	3212.0178
	STD	1.2146	1.4740	2.6514	9.8396	3.5893	7.2071	9.1376	7.6116	15.6195	9.7491	8.7736	11.5878			
Hybrid Quantum Neural Network																
QBN	MEAN	0.0029	0.0055	0.0096	0.0162	0.0248	0.0437	0.0774	0.1498	0.3656	1.1677	9.7629	46.5285	217.2490	1003.2794	4676.9508
	STD	0.0001	0.0003	0.0004	0.0005	0.0007	0.0175	0.0030	0.0029	0.0116	0.0168	0.0978	0.7141			
QPL	MEAN	0.0042	0.0098	0.0135	0.0219	0.0328	0.0484	0.0695	0.1729	0.7119	4.9052	41.7528	213.8433	1057.6980	5150.4466	27499.8175
	STD	0.0006	0.0171	0.0002	0.0112	0.0192	0.0270	0.0156	0.0290	0.0373	0.2127	0.9243	8.0949			
APL	MEAN	0.0057	0.0096	0.0167	0.0272	0.0388	0.0563	0.0861	0.2027	0.7968	3.8129	39.2271	212.5060	1043.6700	4926.9200	22429.9246
	STD	0.0109	0.0002	0.0003	0.0092	0.0072	0.0109	0.0131	0.0184	0.0281	0.2071	1.0127	2.9169			
ASV	MEAN	14.4530	26.6481	38.8279	51.0942	64.5029	74.9574	88.3625	100.9840	112.5433	122.2724	138.2863	189.3105	290.3000	821.8000	3155.0000
	STD	1.2316	3.3369	3.5860	9.2361	7.7340	4.5772	6.6438	13.1945	11.3345	8.8118	8.5446	9.0637			

TABLE III. Benchmarking results – training of quantum and hybrid quantum neural networks.

Quantum Neural Network																
$n_{\textbf{qubits}}$			**2**	**4**	**6**	**8**	**10**	**12**	**14**	**16**	**18**	**20**	**22**	**24**	**26**	**28**	**30**
Simulator	**QBN**	MEAN	0.0006	0.0008	0.0010	0.0014	0.0014	0.0021	0.0030	0.0046	0.0109	0.0295	0.2269	0.9442	4.0763	17.6377	77.0772
		STD	0.0000	0.0002	0.0003	0.0001	0.0004	0.0008	0.0006	0.0007	0.0009	0.0018	0.0135	0.0271	0.0520		
	QPL	MEAN	0.0011	0.0015	0.0019	0.0040	0.0027	0.0032	0.0041	0.0079	0.0249	0.1285	0.9461	4.1994	20.3387	99.3129	390.8168
		STD	0.0003	0.0001	0.0001	0.0169	0.0002	0.0003	0.0003	0.0007	0.0010	0.0183	0.0550	0.1163	1.5112		
	APL	MEAN	0.0011	0.0016	0.0020	0.0025	0.0030	0.0035	0.0044	0.0076	0.0241	0.0975	0.8895	4.2057	18.7627	81.3886	340.6302
		STD	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0003	0.0014	0.0056	0.0069	0.0149	0.1176		
	ASV	MEAN	2.9186	2.9810	2.9658	2.9148	2.9611	2.9631	3.1480	3.0285	3.6932	3.1081	3.1192	4.5938	5.7911	14.1463	52.1173
		STD	0.6806	0.7128	0.6408	0.6332	0.6130	0.6648	1.3843	0.7282	6.0972	0.7290	0.6413	3.8130	1.0294		
QPU	**IonQ Harmony**	MEAN	257.9168	155.6078	455.7227	128.1430	456.8760										
		STD	405.6818	264.2781	493.2422	83.4089	582.5769										
	OQC Lucy	MEAN	3.5553	4.2667	4.8857	5.0052											
		STD	0.5025	0.5352	1.2849	0.9200											
	Rigetti Aspen-M2	MEAN	19.8218	20.3571	21.5349	20.0242	24.5956	27.0217	36.3195	29.0213	30.4019	39.2857	38.8354	39.2298	41.1695	45.7342	50.5362
		STD	3.1108	2.5300	5.8806	2.7064	4.8186	3.3928	9.9247	15.6830	4.8891	4.8322	7.1394	9.9047	9.0877	3.7660	9.3982
	IBMQ Falcon r5.11	MEAN	18.9823	19.9964	19.5158	20.8323	21.5527	21.6090	23.3926	23.9918	25.2614	27.1860	28.9943	30.2249	31.6950		
		STD	0.6948	2.3790	0.5109	0.6333	0.7015	0.4522	0.9337	0.4424	1.1654	0.8637	0.6914	1.1444	0.8600		
Hybrid Quantum Neural Network																	
Simulator	**QBN**	MEAN	0.0006	0.0008	0.0010	0.0014	0.0014	0.0021	0.0030	0.0046	0.0109	0.02952	0.229162	0.95632	4.119108	18.019226	77.074412
		STD	0.0000	0.0002	0.0003	0.0001	0.0004	0.0008	0.0006	0.0007	0.0009	0.001796	0.012569	0.015437	0.092773		
	QPL	MEAN	0.0011	0.0015	0.0019	0.0040	0.0028	0.0034	0.0043	0.0079	0.0270	0.1314	0.9457	4.6933	20.6906	104.8952	466.5611
		STD	0.0002	0.0001	0.0001	0.0167	0.0003	0.0005	0.0004	0.0007	0.0150	0.0074	0.0398	0.1462	2.6173		
	APL	MEAN	0.0011	0.0016	0.0021	0.0025	0.0030	0.0036	0.0045	0.0078	0.0246	0.1001	0.9136	4.2659	18.6986	95.8960	398.8728
		STD	1.0000	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0002	0.0003	0.0017	0.0077	0.0103	0.0187		
	ASV	MEAN	2.7970	2.8320	2.8027	2.7822	2.7476	2.7362	3.4099	2.8901	2.8085	3.0879	2.8273	3.6522	5.3048	14.3117	51.9528
		STD	0.6013	1.2064	0.6225	0.5762	0.4415	0.4935	6.0821	0.7496	0.5778	2.9173	0.5501	0.6063	0.6095		
QPU	**IonQ Harmony**	MEAN	262.3067	232.7949	379.4396	129.2485	455.5415										
		STD	405.5434	342.4183	509.7297	83.8298	575.1551										
	OQC Lucy	MEAN	3.7157	4.5734	4.6802	5.6457											
		STD	0.7375	0.7556	0.9706	1.4995											
	Rigetti Aspen-M2	MEAN	16.8365	20.7644	23.3819	20.5349	24.0114	25.4231	33.9155	27.0000	28.1941	37.1067	44.8568	35.0372	40.1152	45.8019	51.1153
		STD	2.8938	2.6985	6.3815	2.6125	4.6563	3.5133	4.2292	5.8891	3.2389	4.3796	9.6188	3.9946	6.5734	7.5121	7.1591
	IBMQ Falcon r5.11	MEAN	18.6851	19.0753	21.7630	32.3815	22.5955	22.1030	22.6698	23.5265	24.7820	26.7634	28.3862	29.9197	31.7038		
		STD	0.4072	0.5190	5.5041	9.8036	2.5923	0.3981	0.4838	0.5665	0.6743	0.6848	0.4064	0.4930	0.6160		

TABLE IV. Benchmarking results – inference of quantum and hybrid quantum neural networks.

Pure Quantum Training												
$n_{\mathbf{qubits}}$			**2**	**4**	**6**	**8**	**10**	**12**	**14**	**16**	**18**	**20**
Threads	**1**	MEAN	0.0024	0.0054	0.0099	0.0160	0.0248	0.0437	0.0964	0.3243	1.4042	6.6382
		STD	0.0003	0.0002	0.0005	0.0006	0.0007	0.0163	0.0023	0.0021	0.0040	0.0066
	2	MEAN	0.0026	0.0060	0.0118	0.0195	0.0299	0.0456	0.1439	0.2196	1.1028	3.7856
		STD	0.0001	0.0002	0.0002	0.0004	0.0009	0.0016	0.2070	0.0255	0.3134	0.0075
	4	MEAN	0.0027	0.0066	0.0139	0.0211	0.0331	0.0501	0.0866	0.1724	0.5542	2.3383
		STD	0.0001	0.0001	0.0021	0.0009	0.0017	0.0021	0.0026	0.0114	0.0102	0.0400
	8	MEAN	0.0029	0.0090	0.0165	0.0272	0.0406	0.0561	0.0904	0.1636	0.4147	1.5209
		STD	0.0003	0.0008	0.0012	0.0013	0.0020	0.0018	0.0043	0.0122	0.0132	0.0262
	12	MEAN	0.0036	0.0125	0.0236	0.0356	0.0517	0.0716	0.1064	0.1674	0.3927	1.3844
		STD	0.0002	0.0019	0.0006	0.0003	0.0006	0.0009	0.0050	0.0043	0.0061	0.0323
	16	MEAN	0.0037	0.0121	0.0261	0.0431	0.0603	0.0835	0.1104	0.1737	0.3786	1.2194
		STD	0.0001	0.0002	0.0005	0.0008	0.0007	0.0016	0.0078	0.0026	0.0251	0.0373
	20	MEAN	0.0040	0.0130	0.0335	0.0583	0.0711	0.1098	0.1424	0.1930	0.4020	1.2593
		STD	0.0004	0.0008	0.0006	0.0011	0.0008	0.0022	0.0026	0.0036	0.0095	0.0298
	24	MEAN	0.0042	0.0163	0.0355	0.0618	0.0879	0.1229	0.1577	0.2179	0.4339	1.2116
		STD	0.0001	0.0009	0.0004	0.0008	0.0006	0.0047	0.0028	0.0062	0.0158	0.0283
Best thread count			**1**	**1**	**1**	**1**	**1**	**1**	**4**	**8**	**16**	**24**

TABLE V. The performance of QMware basiq native (QBN) when training the QNN quantum network for different numbers of threads.

Pure Quantum Forward												
$n_{\mathbf{qubits}}$			**2**	**4**	**6**	**8**	**10**	**12**	**14**	**16**	**18**	**20**
Threads	**1**	MEAN	0.0008	0.0007	0.0009	0.0011	0.0015	0.0020	0.0034	0.0099	0.0400	0.1687
		STD	0.0001	0.0001	0.0002	0.0003	0.0005	0.0006	0.0008	0.0008	0.0148	0.0087
	2	MEAN	0.0007	0.0008	0.0010	0.0012	0.0030	0.0020	0.0030	0.0068	0.0236	0.0935
		STD	0.0001	0.0002	0.0002	0.0001	0.0002	0.0003	0.0003	0.0004	0.0002	0.0024
	4	MEAN	0.0006	0.0008	0.0011	0.0017	0.0017	0.0021	0.0026	0.0050	0.0150	0.0576
		STD	0.0000	0.0000	0.0000	0.0001	0.0000	0.0002	0.0000	0.0002	0.0004	0.0019
	8	MEAN	0.0006	0.0009	0.0011	0.0018	0.0018	0.0021	0.0026	0.0039	0.0118	0.0389
		STD	0.0000	0.0000	0.0000	0.0001	0.0000	0.0001	0.0000	0.0000	0.0005	0.0040
	12	MEAN	0.0007	0.0013	0.0014	0.0022	0.0022	0.0026	0.0029	0.0046	0.0122	0.0388
		STD	0.0000	0.0000	0.0000	0.0001	0.0000	0.0001	0.0000	0.0003	0.0014	0.0037
	16	MEAN	0.0007	0.0013	0.0016	0.0026	0.0023	0.0029	0.0030	0.0048	0.0111	0.0346
		STD	0.0000	0.0000	0.0000	0.0001	0.0002	0.0001	0.0002	0.0002	0.0009	0.0022
	20	MEAN	0.0008	0.0015	0.0022	0.0033	0.0036	0.0035	0.0039	0.0059	0.0119	0.0356
		STD	0.0000	0.0000	0.0000	0.0001	0.0002	0.0001	0.0001	0.0001	0.0005	0.0018
	24	MEAN	0.0009	0.0016	0.0028	0.0036	0.0036	0.0044	0.0044	0.0063	0.0120	0.0355
		STD	0.0000	0.0000	0.0001	0.0001	0.0001	0.0006	0.0002	0.0001	0.0004	0.0021
Best thread count			**4**	**1**	**1**	**1**	**1**	**1**	**8**	**8**	**16**	**16**

TABLE VI. The performance of QMware basiq native (QBN) when performing inference using the QNN quantum network for different numbers of threads.

Hybrid Quantum Forward												
$n_{\textbf{qubits}}$			**2**	**4**	**6**	**8**	**10**	**12**	**14**	**16**	**18**	**20**
Threads	**1**	MEAN	0.0008	0.0008	0.0010	0.0028	0.0014	0.0021	0.0036	0.0104	0.0419	0.1680
		STD	0.0001	0.0002	0.0003	0.0159	0.0004	0.0008	0.0008	0.0013	0.0168	0.0105
	2	MEAN	0.0007	0.0010	0.0011	0.0014	0.0016	0.0022	0.0331	0.0071	0.0242	0.0932
		STD	0.0001	0.0001	0.0001	0.0002	0.0002	0.0002	0.0076	0.0003	0.0034	0.0017
	4	MEAN	0.0006	0.0010	0.0011	0.0014	0.0021	0.0022	0.0059	0.0052	0.0177	0.0617
		STD	0.0000	0.0000	0.0001	0.0001	0.0002	0.0001	0.0061	0.0002	0.0064	0.0103
	8	MEAN	0.0006	0.0010	0.0012	0.0015	0.0023	0.0024	0.0030	0.0046	0.0118	0.0414
		STD	0.0001	0.0001	0.0000	0.0000	0.0001	0.0001	0.0006	0.0007	0.0011	0.0100
	12	MEAN	0.0007	0.0014	0.0014	0.0017	0.0026	0.0029	0.0035	0.0055	0.0123	0.0363
		STD	0.0000	0.0000	0.0000	0.0000	0.0001	0.0003	0.0002	0.0001	0.0008	0.0046
	16	MEAN	0.0008	0.0015	0.0016	0.0020	0.0030	0.0032	0.0039	0.0055	0.0109	0.0308
		STD	0.0001	0.0000	0.0000	0.0001	0.0001	0.0004	0.0006	0.0002	0.0009	0.0019
	20	MEAN	0.0008	0.0016	0.0023	0.0026	0.0035	0.0038	0.0053	0.0060	0.0121	0.0312
		STD	0.0000	0.0001	0.0002	0.0000	0.0001	0.0002	0.0003	0.0002	0.0004	0.0019
	24	MEAN	0.0009	0.0019	0.0031	0.0034	0.0044	0.0040	0.0057	0.0062	0.0118	0.0295
		STD	0.0000	0.0002	0.0000	0.0001	0.0001	0.0002	0.0002	0.0001	0.0002	0.0018
Best thread count			**4**	**1**	**1**	**4**	**1**	**1**	**8**	**8**	**16**	**24**

TABLE VII. The performance of QMware basiq native (QBN) when performing inference using the HQNN for different numbers of threads.

Hybrid Quantum Training												
$n_{\textbf{qubits}}$			**2**	**4**	**6**	**8**	**10**	**12**	**14**	**16**	**18**	**20**
Threads	**1**	MEAN	0.0008	0.0007	0.0009	0.0011	0.0015	0.0020	0.0034	0.0099	0.0400	0.1687
		STD	0.0001	0.0001	0.0002	0.0003	0.0005	0.0006	0.0008	0.0008	0.0148	0.0087
	2	MEAN	0.0007	0.0008	0.0010	0.0012	0.0030	0.0020	0.0030	0.0068	0.0236	0.0935
		STD	0.0001	0.0002	0.0002	0.0001	0.0002	0.0003	0.0003	0.0004	0.0002	0.0024
	4	MEAN	0.0006	0.0008	0.0011	0.0017	0.0017	0.0021	0.0026	0.0050	0.0150	0.0576
		STD	0.0000	0.0000	0.0000	0.0001	0.0000	0.0002	0.0000	0.0002	0.0004	0.0019
	8	MEAN	0.0006	0.0009	0.0011	0.0018	0.0018	0.0021	0.0026	0.0039	0.0118	0.0389
		STD	0.0000	0.0000	0.0000	0.0001	0.0000	0.0001	0.0000	0.0000	0.0005	0.0040
	12	MEAN	0.0007	0.0013	0.0014	0.0022	0.0022	0.0026	0.0029	0.0046	0.0122	0.0388
		STD	0.0000	0.0000	0.0000	0.0001	0.0000	0.0001	0.0000	0.0003	0.0014	0.0037
	16	MEAN	0.0007	0.0013	0.0016	0.0026	0.0023	0.0029	0.0030	0.0048	0.0111	0.0346
		STD	0.0000	0.0000	0.0000	0.0001	0.0002	0.0001	0.0002	0.0002	0.0009	0.0022
	20	MEAN	0.0008	0.0015	0.0022	0.0033	0.0036	0.0035	0.0039	0.0059	0.0119	0.0356
		STD	0.0000	0.0000	0.0000	0.0001	0.0002	0.0001	0.0001	0.0001	0.0005	0.0018
	24	MEAN	0.0009	0.0016	0.0028	0.0036	0.0036	0.0044	0.0044	0.0063	0.0120	0.0355
		STD	0.0000	0.0000	0.0001	0.0001	0.0001	0.0006	0.0002	0.0001	0.0004	0.0021
Best thread count			**4**	**1**	**1**	**1**	**1**	**1**	**8**	**8**	**16**	**16**

TABLE VIII. The performance of QMware basiq native (QBN) when training the hybrid quantum network for different numbers of threads.

Accuracy, %										
n_{qubits}		2	4	6	8	10	12	14	16	18
IonQ Harmony	MEAN	99.76	99.73	99.52	99.19	99.12				
	STD	0.20	0.17	0.12	0.27	0.30				
Rigetti Aspen-M2	MEAN	88.77	5.47	0.67	0.15	0.04	0.02	0.00	0.00	0.00
	STD	6.81	0.75	2.65	0.12	0.07	0.04	0.00	0.00	0.00
OQC Lucy	MEAN	93.76	67.87	63.66	57.64					
	STD	0.69	1.98	1.59	0.93					
IBMQ Falcon r5.11	MEAN	93.23	54.40	2.97	1.40	0.13	0.03	0.00		
	STD	0.42	4.20	0.50	0.22	0.05	0.05	0.00		

TABLE IX. QPU accuracies

Times, seconds										
n_{qubits}		2	4	6	8	10	12	14	16	18
IonQ Harmony	MEAN	350.2784	80.0766	339.8358	17.9963	73.2473				
	STD	501.8712	59.0565	630.4188	7.0536	102.1677				
Rigetti Aspen-M2	MEAN	58.7262	52.1269	60.1630	64.3308	58.6161	85.4907	85.7971	81.8377	99.0498
	STD	35.1338	21.8146	17.8216	16.9935	20.6759	26.5828	22.4378	33.9639	19.8638
OQC Lucy	MEAN	3.4535	3.3095	3.3682	3.4450					
	STD	0.4990	0.2007	0.2636	0.3664					
IBMQ Falcon r5.11	MEAN	43.1557	20.1590	17.6970	18.2476	21.6717	18.6053	18.9948		
	STD	48.3393	4.9506	0.8301	0.8283	8.4172	0.3280	0.9642		

TABLE X. QPUs runtimes for the accuracy test

$\mathbf{n_{qubits}}$			**32**	**34**	**36**	**38**	**40**
QNN	**QBN**		304.1323	1343.5551	5952.5783	28893.5688	48970.1862
	ASV		237.3554	986.5651			
	Rigetti Aspen-M2	MEAN	51.9887	52.1990	59.0934	60.9397	73.9763
		STD	8.9592	7.9419	11.5971	7.4355	11.4524
HQNN	**QBN**		270.6306	1244.3166	3450.6118	18630.4505	47273.2937
	ASV		323.9802	1414.4067			
	Rigetti Aspen-M2	MEAN	49.9818	52.4805	61.9770	64.4956	67.4168
		STD	6.9296	7.0688	17.2564	7.6582	7.9843

TABLE XI. Inference runtimes beyond 30 qubits