Benchmarking Simulated and Physical Quantum Processing Units Using Quantum and Hybrid Algorithms

Mohammad Kordzanganeh, Markus Buchberger, Basil Kyriacou, Maxim Povolotskii, Wilhelm Fischer, Andrii Kurkin, Wilfrid Somogyi, Asel Sagingalieva, Markus Pfïtsch, and Alexey Melnikov*

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. 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 simulator 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 Amazon Web Service State-Vector Simulator 1 offers a runtime advantage for larger circuits, up to the maximum 34 qubits. Beyond this limit, QMware can 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 qubits. However, the high financial cost of physical quantum processing units presents a serious barrier to practical use. Moreover, only IonQ’s Harmony quantum device 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.

1. 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) are quantum machine learning (QML) algorithms[1–4] that 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.[5–12] 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.[13,14]

Unlike their classical counterparts, QNNs are able to learn a generalized model of a dataset from a substantially smaller training set[15–17] and typically have the potential to do so with polynomially or exponentially simpler models.[18–20] Thus, they provide a promising opportunity to subvert the scaling problem encountered in classical machine learning.[5,21–27] which presents a serious challenge for data-intensive problems that are increasingly bottle-necked by hardware limitations.[28–30]

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,[31] 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 behavior. 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, Amazon Web Service State-Vector Simulator 1 (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. The Python implementation code for the benchmarks presented in this study is available in ref. [32].

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,[33] as well as software backends such as the IBM Qiskit machine learning suite and the QuLacs package.[34,35] It also includes QPUs such as IBM’s Eagle processor,[36] Honeywell’s system model H1,[37] and Google’s Sycamore processor.[38] 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. [39, 40] as well as metric-specific tests such as refs. [41–47].

The work is organized in five sections. Section 2 describes the methodology, including the benchmark algorithms, the hardware and software tested, as well as the results of the runtime benchmark. Section 3 details the cost of executing the proposed quantum circuits on various QPUs and compares their fidelity. Section 4 discusses the execution of large quantum circuit, with as many as 40 qubits, and the implementation of multi-threading in simulators. Finally, Section 4.3 provides a summary of the findings, and outlines key challenges for the quantum computing industry. 

2. 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 $2^n$. 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.[46] It is also the recommended quantum simulation library for the AWS Braket computing service, which provides a performance-optimized version of the PennyLane library.[45] 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.[48] 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.[50] 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.

2.1. Methodology

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 schematic of this methodology is presented in Figure 1.

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 PennyLane lightning.qubit (APL) benchmark represents single-core calculations on both hardware services.
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, QQC, Rigetti, IBM, and QMware.

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, \( n_q = 2n_d \) (see Section 2.2 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. 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.

### 2.2. Benchmark Dataset

The dataset for the benchmarks presented here is an \( n \)-dimensional abstracted version of the well-documented 2D sklearn.datasets.make_circles for binary classification tasks. It consists of points sampled from two concentric circular shells, distributed randomly about two nominal mean radii, \( r_{\text{inner}} < r_{\text{outer}} \). 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 ref. [53] 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 \( \sigma = 1 \)

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

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

\[
\vec{x} = r \frac{\vec{v}}{r} + \vec{\rho}, \quad \rho_i \in \mathcal{N}(0, \sigma^2)
\]

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 \( r_{\text{outer}} = 1.0 \) is used to create points in the outer shell and \( r_{\text{inner}} = 0.2 \) to create points in the inner shell, with \( \sigma = 0.3 \) 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 1D, 2D and, 3D datasets are shown in Figure 2.

### 2.3. 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 \( \alpha = 0.3 \).

#### 2.3.1. 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...
ref. [15]. 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 $R_z$ 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 ($\sigma_z$) 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.\textsuperscript{55,56}

2.3.2. 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 Section 2.3.1, 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 \([p_L, 40, 1]\), with ReLU and Sigmoid activation functions applied to the input and hidden layer, respectively.\textsuperscript{[57]} 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 r3.11 whose classical part was executed on QMware HQC4020.

2.4. Results

This section presents the results of the benchmark methodology outlined in Section 2.1. The results of the benchmark are illustrated in Figure 2. A table of these results is also given in Appendix B, Tables B1 and B2. 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.\textsuperscript{[58]}

2.4.1. 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 Table 1 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 \(t_{QPL}/t_{APL} = 0.90 \pm 0.17\). When benchmarking with the HQNN the average relative runtime is \(t_{QPL}/t_{APL} = 0.95 \pm 0.16\).

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 \(t_{QBN}/t_{QPL} = 0.56 \pm 0.32\) over QPL in the QNN benchmark, and \(t_{QBN}/t_{QPL} = 0.54 \pm 0.30\) 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 \(t_{QBN}/t_{QPL} = 0.22 \pm 0.04\) is obtained for the QNNs and \(t_{QBN}/t_{QPL} = 0.21 \pm 0.02\) 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 \(t_{QBN}/t_{ASV} = 1.35 \pm 0.11\). 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 \(t_{QBN}/t_{ASV} = 0.00041 \pm 0.00025\) for circuits with 14 qubits or less, \(t_{QBN}/t_{ASV} = 0.0049 \pm 0.0035\) for circuits with 16 to 20 qubits, and \(t_{QBN}/t_{ASV} = 0.36 \pm 0.29\) for circuits with 22 to 26 qubits.

2.4.2. 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 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 \(t_{QPL}/t_{APL} = 1.04 \pm 0.10\) for QNNs and \(t_{QPL}/t_{APL} = 1.03 \pm 0.22\) for HQNNs. For larger circuits, APL performs marginally better than QPL with \(t_{QPL}/t_{APL} = 1.11 \pm 0.10\) and \(t_{QPL}/t_{APL} = 1.12 \pm 0.09\) for QNNs and HQNNs, respectively.

Crucially, the inference tests include a benchmark of physical QPUs, as listed in Table 2. 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 result, 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

### Table 1. The abbreviations associated with each hardware, SDK, and backend.

| 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 |

---

Adv. Quantum Technol. 2023, 6, 2300043 2300043 (5 of 16) © 2023 Terra Quantum AG and QMware AG. Advanced Quantum Technologies published by Wiley-VCH GmbH
Table 2. Quantum processing units used in the hardware benchmarking tests, their qubit counts, and native gates.

| 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      |

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 \( t_{\text{ASV}} / t_{\text{M-2}} = 1.03 \), \( t_{\text{QBN}} / t_{\text{M-2}} = 1.53 \) for the QNN and \( t_{\text{ASV}} / t_{\text{M-2}} = 1.02 \), \( t_{\text{QBN}} / t_{\text{M-2}} = 1.51 \) for the HQNN. For smaller circuits sizes OQC’s Lucy produces runtimes that are faster by a factor of \( t_{\text{Lucy}} / t_{\text{M-2}} = 0.22 \pm 0.03 \) compared to Rigetti’s Aspen M-2. In contrast, the runtime for IonQ’s Harmony QPU on average a factor \( t_{\text{Harmony}} / t_{\text{M-2}} = 13.5 \pm 5.2 \) 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 \( t_{\text{ASV}} / t_{\text{M-2}} = 0.87 \pm 0.18 \).

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. Specifically, the average percentage standard deviation across the 10 repeat measurements is on the order of 125%, 16%, 21%, and 7% for IonQ, Aspen M-2, Lucy, and Falcon r5.11, respectively.

### 3. Accuracy and Cost Analysis

#### 3.1. Cost Evaluation

As described in Section 1 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 Sections 2.3.1 and 2.3.2, 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. Figure 3b 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 ≈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. It is worth clarifying that the training in this benchmark treats every quantum evaluation as a new circuit on which many shots can be executed. The IBM Qiskit Runtime offers an alternative method: where the quantum computer scientist can set up the quantum circuit once for a large initial cost and then all the circuit executions in that epoch would use the same setup but with varying parameters depending on the dataset and the parameter-shift rule. Assuming a circuit set-up that takes \( t_1 = 5 \) s and a per-shot runtime of \( t_2 = 250 \mu s \), the cost of training the same circuit could be dramatically reduced from tens of millions of USD to 212 800 USD. The multi-parameter, multi-circuit functionality is specific to the IBM Cloud and thus the authors decided to set up every circuit in all cases in the interest of fairness.

In contrast to the high cost of training, 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). A typical forward-pass of a QNN can be executed at a cost of ≈1 USD for QPUs accessed through AWS Braket, or between 10–40 USD for QPUs accessed through Qiskit runtime.

#### 3.2. 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 \( \pi/4 \), 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, that is, \( <00⋯0|\psi\rangle = 0 \). 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 \( 2^n - 1 \) 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
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 $\approx 16 \times 2^n$ bytes of memory. This equates to $\approx 16$ 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 vanish exponentially as the number of qubits increases. This can present a serious obstacle to training a QML model with a large number of qubits unless a variety of mitigation methods are employed. Solving the barren plateau problem is crucial to achieving highly expressive models that could outperform classical machine learning models 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. Figure 4d illustrates the exponential increase in memory usage for a range of circuit sizes up to 40 qubits, for both single and double precision representations.

Table B9 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 h 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 $t_{\text{ASV}} / t_{\text{QBN}} = 1.34 \pm 0.09$ for circuits with 28 to 34 qubits. In the case of HQNNs there is no clear advantage, with $t_{\text{ASV}} / t_{\text{QBN}} = 0.95 \pm 0.22$.

4.2. 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. Figure 4a illustrates a cross-sectional study of runtime for circuits with up to 20 qubits with multi-threading across as many as 24 threads. In Figure 4b 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,[56] 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.

### 4.3. Conclusion

This work presents a comprehensive study of various quantum computing platforms, using both simulated and physical quantum processing units. The results presented in Section 2.4 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 $0.56 \pm 0.32$ relative to the PennyLane library for QNNs and $0.54 \pm 0.30$ for HQQNs.

The QMware HQC4020 hardware benchmarked against AWS ml.m5.24xlarge achieves a comparable relative runtime of $0.90 \pm 0.17$ for QNNs and $0.95 \pm 0.16$ for HQQNs. 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 HQQNs. 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 $\approx 1$ min, 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 Section 3.2 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.

Appendix A: Current Benchmarking Landscape

As described in ref. [64], quantum benchmarking faces many more challenges than its classical counterparts. After all, there is no singular “quantum technology.” Quantum devices are made from different materials and used for vastly different purposes. This makes it difficult to predict how the field will progress, as each type of technology develops breakthroughs at different rates. Additionally, quantum devices are ultimately compared not only to other quantum computers but also to the performance of classical devices, itself an ever-moving target.

Given the multitude of challenges, it is important to question how any benchmark result compares to others. In this appendix, a non-exhaustive list of key takeaways from other benchmarking papers is presented to better address where this paper sits in the wider landscape. The year of each paper’s publication can be viewed in Figure A1. For more information on the current state of quantum benchmarking, ref. [65] provides an excellent summary of many benchmarking papers. In that paper, Wang et al. separate quantum benchmarks into three classes: physical, aggregative, and application-level.

Physical benchmarks depend on hardware specificity. These tasks give a measure of the engineering capabilities and limitations of quantum devices, such as the impacts of decoherence and noise, which have been established as major speedbumps in the development of fully-functional circuits, such as the impacts of decoherence and noise, which have been established as major speedbumps in the development of fully-functional quantum devices. Examples of physical benchmarking papers include ref. [39], a measurement of qubit coherences and gate-based fidelities of IonQ’s trapped Ytterbium qubits, and ref. [67], which measures T1 and T2 coherence times as well as the CCNOT gate and the adjoint Quantum Fourier Transform for a 20-qubit IBM device.

Aggregated benchmarks measure how well multiple qubits and gates work together. Perhaps the most widely used of these hardware-agnostic metrics is quantum volume. Introduced by IBM in ref. [68], quantum volume measures the maximum circuit width (number of qubits) and depth (layers of gates) that a device can run such that the two numbers are equal and the fidelity is maintained above a certain threshold. Given a square circuit of width and depth size $N$, its quantum volume is defined as the exponentiation of the side length of this square, $2^N$, and the side length itself is sometimes referred to as the number of algorithmic qubits. For example, a quantum device that can attain high fidelity with 4 qubits and a circuit depth of 4 has a quantum volume of $2^4 = 16$ and is said to have 4 algorithmic qubits.

![Figure A1](https://example.com/figure-a1.png)

**Figure A1.** Timeline of key benchmarking papers considered in Appendix: Current Benchmarking Landscape. The bracketed numbers represent the citations of the referenced papers.
Table B1. Benchmarking results – training of quantum and hybrid quantum neural networks.

| n_qubits | Quantum neural network | Hybrid quantum neural network |
|----------|------------------------|------------------------------|
| 2        | QBN MEAN 0.0024        | ASV MEAN 14.0362             |
|          | STD 0.0003             |                              |
| 4        | QPL MEAN 0.0039        |                              |
|          | STD 0.0006             |                              |
| 6        | APL MEAN 0.0058        |                              |
|          | STD 0.0109             |                              |
| 8        | ASV MEAN 14.0362       |                              |
|          | STD 1.2146             |                              |
| 10       | QBN MEAN 0.0024        |                              |
|          | STD 0.0003             |                              |
| 12       | QPL MEAN 0.0039        |                              |
|          | STD 0.0006             |                              |
| 14       | APL MEAN 0.0058        |                              |
|          | STD 0.0109             |                              |
| 16       | ASV MEAN 14.0362       |                              |
|          | STD 1.2146             |                              |
| 18       | QBN MEAN 0.0024        |                              |
|          | STD 0.0003             |                              |
| 20       | QPL MEAN 0.0039        |                              |
|          | STD 0.0006             |                              |
| 22       | APL MEAN 0.0058        |                              |
|          | STD 0.0109             |                              |
| 24       | ASV MEAN 14.0362       |                              |
|          | STD 1.2146             |                              |
| 26       | QBN MEAN 0.0024        |                              |
|          | STD 0.0003             |                              |
| 28       | QPL MEAN 0.0039        |                              |
|          | STD 0.0006             |                              |
| 30       | APL MEAN 0.0058        |                              |
|          | STD 0.0109             |                              |
Table B2. Benchmarking results – inference of quantum and hybrid quantum neural networks.

| Simulator | Quantum neural network | Hybrid quantum neural network |
|-----------|-----------------------|-----------------------------|
| QN        | Mean                  | Mean                        |
| QPN       | Mean                  | Mean                        |
| QPL       | Mean                  | Mean                        |
| QPL       | Mean                  | Mean                        |
| APL       | Mean                  | Mean                        |
| ASV       | Mean                  | Mean                        |
| QPU       | Mean                  | Mean                        |
| OQCL       | Mean                  | Mean                        |
| Rigetti Aspen-M2 | Mean                  | Mean                        |
| IBMQ Falcon, S1 | Mean                  | Mean                        |

For each simulator, the table lists the mean and standard deviation (STD) of the inference times for both quantum and hybrid quantum neural networks.
Table B3. The performance of QMware basic native (QBN) when training the QNN quantum network for different numbers of threads.

| $n_{\text{qubits}}$ | 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 |
| Threads 2 MEAN      | 0.0026 | 0.0060 | 0.0118 | 0.0195 | 0.0299 | 0.0456 | 0.1439 | 0.2196 | 1.1028 | 3.7586 |
| Threads 4 MEAN      | 0.0027 | 0.0066 | 0.0139 | 0.0211 | 0.0311 | 0.0551 | 0.0866 | 0.1724 | 0.5542 | 2.3383 |
| Threads 8 MEAN      | 0.0029 | 0.0090 | 0.0165 | 0.0272 | 0.0399 | 0.0531 | 0.1461 | 0.2196 | 1.1028 | 3.7586 |
| Threads 12 MEAN     | 0.0036 | 0.0125 | 0.0236 | 0.0356 | 0.0517 | 0.0716 | 0.1064 | 0.1674 | 0.3927 | 1.3844 |
| Threads 16 MEAN     | 0.0037 | 0.0121 | 0.0261 | 0.0431 | 0.0603 | 0.0835 | 0.1104 | 0.1737 | 0.3786 | 1.2194 |
| Threads 20 MEAN     | 0.0040 | 0.0130 | 0.0335 | 0.0583 | 0.0711 | 0.1098 | 0.1424 | 0.1930 | 0.4202 | 1.2595 |
| Threads 24 MEAN     | 0.0042 | 0.0163 | 0.0355 | 0.0618 | 0.0879 | 0.1229 | 0.1577 | 0.2179 | 0.4339 | 1.2116 |

Best thread count 1 1 1 1 1 4 8 16 24

Table B4. The performance of QMware basic native (QBN) when performing inference using the QNN quantum network for different numbers of threads.

| $n_{\text{qubits}}$ | 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 |
| Threads 2 MEAN      | 0.0007 | 0.0008 | 0.0010 | 0.0012 | 0.0030 | 0.0020 | 0.0030 | 0.0068 | 0.0236 | 0.0935 |
| Threads 4 MEAN      | 0.0006 | 0.0008 | 0.0011 | 0.0017 | 0.0021 | 0.0026 | 0.0050 | 0.0150 | 0.0576 |
| Threads 8 MEAN      | 0.0006 | 0.0009 | 0.0011 | 0.0018 | 0.0021 | 0.0026 | 0.0039 | 0.0118 | 0.0389 |
| Threads 12 MEAN     | 0.0007 | 0.0013 | 0.0014 | 0.0022 | 0.0026 | 0.0029 | 0.0046 | 0.0122 | 0.0388 |
| Threads 16 MEAN     | 0.0007 | 0.0013 | 0.0016 | 0.0026 | 0.0023 | 0.0029 | 0.0030 | 0.0048 | 0.0111 | 0.0346 |
| Threads 20 MEAN     | 0.0008 | 0.0015 | 0.0022 | 0.0033 | 0.0036 | 0.0033 | 0.0059 | 0.0119 | 0.0356 |
| Threads 24 MEAN     | 0.0009 | 0.0016 | 0.0028 | 0.0036 | 0.0044 | 0.0044 | 0.0063 | 0.0120 | 0.0355 |

Best thread count 4 1 1 1 1 1 1 8 8 16 16

shallow circuits (larger circuit width) would be better suited, such as state preparation and IQP circuits respectively. While ref. [42] utilizes volumetric benchmarks as a backdrop to measure a suite of quantum algorithms, such as the quantum Fourier transform and Grover’s search. These algorithms are tested many times at different depths and widths. Consistently, as the tasks increase in size and move farther away from the center of the quantum volume square, the fidelity drops. In this way, the relationship between a device’s quantum volume and the fidelity of quantum operations is readily apparent through visually accessible figures that show when the devices begin to fail. The use of visual assessments allows for an intuitive understanding of benchmarking and has become more standard. For example, ref. [70] tests 21 quantum devices with algorithms that were not only chosen for performance but also specifically chosen with visual accessibility in mind.

Finally, benchmarks from the field of quantum annealing, which utilizes the similarity of a physical Ising model and quadratic unconstrained binary optimization (QUBO) to solve combinatorial optimization tasks, are necessary for annealing devices. Ref. [71] showed that specific instances...
Table B5. The performance of QMware basis Qnative (QBN) when performing inference using the HQNN for different numbers of threads.

| Threads | \(n_{\text{qubits}}\) | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
|---------|-----------------|---|---|---|---|----|----|----|----|----|----|
| STD     | MEAN            | 0.0008 | 0.0007 | 0.0009 | 0.0011 | 0.0015 | 0.0020 | 0.0034 | 0.0099 | 0.0400 | 0.1687 |
| STD     | MEAN            | 0.0001 | 0.0001 | 0.0002 | 0.0003 | 0.0005 | 0.0006 | 0.0008 | 0.0008 | 0.0148 | 0.0087 |
| STD     | MEAN            | 0.0007 | 0.0008 | 0.0010 | 0.0012 | 0.0030 | 0.0020 | 0.0030 | 0.0068 | 0.0236 | 0.0935 |
| STD     | MEAN            | 0.0001 | 0.0002 | 0.0002 | 0.0001 | 0.0002 | 0.0003 | 0.0003 | 0.0004 | 0.0002 | 0.0024 |
| STD     | MEAN            | 0.0006 | 0.0008 | 0.0011 | 0.0017 | 0.0017 | 0.0021 | 0.0026 | 0.0050 | 0.0150 | 0.0576 |
| STD     | MEAN            | 0.0000 | 0.0000 | 0.0000 | 0.0001 | 0.0001 | 0.0002 | 0.0002 | 0.0002 | 0.0002 | 0.0002 |
| STD     | MEAN            | 0.0009 | 0.0019 | 0.0031 | 0.0034 | 0.0044 | 0.0040 | 0.0057 | 0.0062 | 0.0118 | 0.0295 |
| STD     | MEAN            | 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 1 1 1 1 1 8 8 16 24

Table B6. The performance of QMware basis Qnative (QBN) when training the hybrid quantum network for different numbers of threads.

| Threads | \(n_{\text{qubits}}\) | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
|---------|-----------------|---|---|---|---|----|----|----|----|----|----|
| STD     | MEAN            | 0.0007 | 0.0013 | 0.0014 | 0.0022 | 0.0026 | 0.0029 | 0.0046 | 0.0122 | 0.0388 | 0.0040 |
| STD     | MEAN            | 0.0000 | 0.0000 | 0.0000 | 0.0001 | 0.0001 | 0.0001 | 0.0000 | 0.0000 | 0.0003 | 0.0037 |
| STD     | MEAN            | 0.0007 | 0.0013 | 0.0016 | 0.0026 | 0.0023 | 0.0029 | 0.0030 | 0.0048 | 0.0111 | 0.0346 |
| STD     | MEAN            | 0.0000 | 0.0000 | 0.0000 | 0.0001 | 0.0001 | 0.0002 | 0.0002 | 0.0002 | 0.0002 | 0.0022 |
| STD     | MEAN            | 0.0008 | 0.0015 | 0.0022 | 0.0033 | 0.0036 | 0.0039 | 0.0059 | 0.0119 | 0.0356 | 0.0123 |
| STD     | MEAN            | 0.0000 | 0.0000 | 0.0000 | 0.0001 | 0.0001 | 0.0002 | 0.0001 | 0.0001 | 0.0002 | 0.0018 |

Best thread count
4 1 1 1 1 1 1 1 8 8 16 16
d of quantum annealing on DWave’s advantage were able to produce solutions within 0.5% of the best-known solution in a fraction of the time of the fastest classical algorithms. It should be noted this is not a fundamental speedup, nevertheless, order-of-magnitude differences in execution time obviously have substantial industry potential.

Application-based benchmarks span a wide field of practical industry tasks as well as theoretical problems tailored to comparing devices. Ultimately, while there have been many proposed benchmarks, they share a common theme in that they attempt to answer the degree to which a given quantum device can perform industry tasks and effectively communicate results. With this in mind, Figure A2 provides an inexact categorization of these benchmarks based on the specificity of the metrics—that is, the generality of a given benchmark with respect to the hardware required and algorithms available—as well as its practicality on current noisy quantum computers.

This paper’s testing of supervised learning on various devices serves as an application-based benchmark that has yielded preliminary results on the role of neural networks in quantum machine learning. This
document tests a range of real and simulated quantum devices, comparing their time and accuracy performance on a classification task. The future role that quantum machines will play in high-level neural networks and deep learning remains to be determined, but these results demonstrate practical effects of quantum hardware as well as quantum-inspired classical hardware.

**Appendix B: Raw Numerical Results**

In this section, the raw results of the benchmarks are provided in tabular view. Table B1 demonstrates the data obtained during the training process, whereas Table B2 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. Tables B3–B6 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, Table B7 provides the numerical results for the QPU accuracy tests, whereas the times obtained are shown in Table B8. These runtimes are largely in agreement with those in Table B1. However, it is worth noting that these circuits are twice as deep as the ones used in runtime benchmarks as explained in Section 3.2. Table B9 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

### Table B7. QPU accuracies.

| $n_{\text{qubits}}$ | 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    |
|                      | STD     | 6.81    | 0.75    | 2.63    | 0.12    | 0.07    | 0.04    | 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    | 0.00    |
|                      | STD     | 0.42    | 4.20    | 0.50    | 0.22    | 0.05    | 0.05    | 0.00    | 0.00    |

### Table B8. QPUs runtimes for the accuracy test.

| $n_{\text{qubits}}$ | 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 | 85.4007 | 85.7971 | 81.8377 |
|                      | STD     | 35.1338 | 21.8146 | 17.8216 | 16.9935 | 20.6759 | 26.5828 | 22.4378 | 33.9639 |
| 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 B9. Inference runtimes beyond 30 qubits.

| $n_{\text{qubits}}$ | 32      | 34      | 36      | 38      | 40      |
|----------------------|---------|---------|---------|---------|---------|
| QNN                  | 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 B10. Cost of executing inference and training runs on QPUs.

| n_qubits | IBM QMware | Rigetti Aspen-M2 | OQC Lucy | IBM Falcon r5.11 |
|----------|------------|------------------|----------|-----------------|
|          | Inference  | Inference        | Inference| Inference        |
| 2        | 10.30      | 0.34             | 0.34     | 16.21           |
| 4        | 10.30      | 0.34             | 0.34     | 16.29           |
| 6        | 10.30      | 0.34             | 0.34     | 17.11           |
| 8        | 10.30      | 0.34             | 0.34     | 18.67           |
| 10       | 10.30      | 0.34             | 0.34     | 19.90           |
| 12       | 10.30      | 0.34             | 0.34     | 20.22           |
| 14       | 10.30      | 0.34             | 0.34     | 21.57           |
| 16       | 10.30      | 0.34             | 0.34     | 23.57           |
| 18       | 10.30      | 0.34             | 0.34     | 24.97           |
| 20       | 10.30      | 0.34             | 0.34     | 28.00           |
| 22       | 10.30      | 0.34             | 0.34     | 30.54           |
| 24       | 10.30      | 0.34             | 0.34     | 32.83           |
| 26       | 10.30      | 0.34             | 0.34     | 35.04           |

The data that support the findings of this study are included in the article.

Data Availability Statement
The data that support the findings of this study are included in the article.

Keywords
hardware and software benchmarking, hybrid quantum-classical algorithms, quantum algorithms, quantum processing units, simulated quantum processing units

Conflict of Interest
The authors declare no conflict of interest.

Author Contributions
M.K., M.B., Maxim P., W.F., A.K., W.S., and A.S. wrote the benchmarking Python code. B.K. reviewed the existing benchmarking literature, wrote the Appendix, and automated the benchmarking Python code. M.K., Maxim P., and A.S. performed benchmarking of simulated quantum processing units. M.K., A.K., and W.S. performed benchmarking of physical quantum processing units. A.M., M.K., W.S., and A.S. analyzed the results. M.B. and W.F. set up the QMware simulator, and prepared the basic SDK. M.B. monitored the resource utilization on the QMware simulator, and analyzed the multithreading efficiency. W.S. and M.K., and A.M. wrote the initial version of the main text. A.S. and A.M. prepared the figures. All authors have contributed to improving the manuscript, read, and agreed to the final version of the manuscript. Markus P. and A.M. performed project administration and supervision.

Received: February 22, 2023
Revised: May 7, 2023
Published online: June 14, 2023

[1] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, S. Lloyd, Nature 2017, 549, 195.
