AFRL-RI-RS-TR-2022-132

Q-MEANS: QUANTUM ADVANTAGE FOR CLUSTERING AND CLASSIFICATION

QC WARE, CORP.

SEPTEMBER 2022

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the 88th ABW, Wright-Patterson AFB Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2022-132 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
PAUL M. ALSING (signed for:)
JON M. MAGGIOLINO
Work Unit Manager

/ S /
GREGORY J. HADYNSKI (signed for:)
LAUREN M. HUIE-SEVERSKY, PhD
Technical Advisor, Computing and Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
SEPTEMBER 2022	FINAL TECHNICAL REPORT	MARCH 2020	MARCH 2022

4. TITLE AND SUBTITLE

Q-MEANS: QUANTUM ADVANTAGE FOR CLUSTERING AND CLASSIFICATION

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
FA8750-20-C-0503	N/A	62788F

5d. PROJECT NUMBER	5e. TASK NUMBER	5f. WORK UNIT NUMBER
		R2X3

6. AUTHOR(S)

Victor Putz

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8.PERFORMING ORGANIZATIONREPORT NUMBER
QC Ware, Corp. 195 Page Mill Road, Suite 113 Palo Alto CA 94306	

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10.SPONSOR/MONITOR'S ACRONYM(S)	11.SPONSOR/MONITOR'S REPORT NUMBER(S)
Air Force Research Laboratory/RITQ 525 Brooks Road Rome NY 13441-4505	AFRL/RI	AFRL-RI-RS-TR-2022-132

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for Public Release; Distribution Unlimited. PA# AFRL-2022-4225
Date Cleared: 06 SEPTEMBER 2022

13. **SUPPLEMENTARY NOTES**

14. ABSTRACT

This project developed a quantum algorithm for clustering, similar to k-means, but that performs the distance estimation aspect of the clustering algorithm in a much more efficient manner, thus promising a potential speedup once quantum computing hardware matures to the point of being able to implement this quantum algorithm. This milestone report benchmarked performance of the quantum circuit used in the Q-Means clustering algorithm on classical simulators and current quantum hardware.

15. SUBJECT TERMS

Quantum computing, clustering algorithm, q-means, flight path classification

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
U	U	U	SAR	49

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
JON M. MAGGIOLINO	N/A

PREVIOUS EDITION IS OBSOLETE.

STANDARD FORM 298 (REV. 5/2020)
Prescribed by ANSI Std. Z39.18

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Q-Means: Quantum Advantage for Clustering and Classification

1. SUMMARY

This effort develops a quantum clustering algorithm called Q-Means that in the future should accelerate United States Air Force (USAF) analysis efforts. The USAF relies increasingly on computational data analysis, a requirement that cuts across many operational fields: signal and photo analysis, supervised and unsupervised learning, pattern recognition in intelligence reports, and much more. One of the fundamental algorithms used in such analysis is k-means, an algorithm which clusters datapoints into groups of similar features. This is fundamental to grouping images, separating audio signals, identifying groups of communications patterns, classifying objects and actions in motion sensors, creating profiles from observations of online activity, and many other sensor data processing purposes.

During the project, we developed several technical innovations to advance the state of the art. First we developed a scheme to represent flight paths for various air vehicles—military, commercial, manned, unmanned—suitable for unsupervised classification. Due to a lack of available data from real and diverse flight paths, we created simulated flight path data, and then used our schema to extract appropriate data features for the flight paths to use in the clustering algorithm. The Q-Means quantum clustering algorithm was built around a fundamental quantum algorithm for estimating distances between vectors. To improve the performance of the quantum computer used in the computations, we implemented error reduction techniques to improve the results. And we developed a software toolset to run the quantum algorithm and flight path data on actual quantum computing hardware commercially available via the cloud.

This study accomplished multiple technical objectives. First, we tested the Q-Means algorithm on synthetic data for diverse flight paths, showing that simulations of quantum computing match theoretical expectations. Then we compared theoretical and simulated performance of Q-Means to show scale of problem size where quantum computing would deliver advantage over classical, indicating that flight paths of at least hundreds of data points in time would be needed before quantum computing would show an advantage over classical. We also measured the performance of Q-Means on actual quantum hardware (IonQ), analyzing performance versus flight path data size and QC hardware noise. We identify additional quantum algorithmic enhancements that should improve theoretical performance. And finally, we identify improvements in the software workflow with cloud-based quantum computers that should yield better results from the hardware, especially for quantum computers accessed remotely via the cloud. In all, this study of the Q-Means algorithm on quantum computers resulted in the first application of quantum computing to airspace situational awareness.

2. INTRODUCTION

Quantum technologies promise to revolutionize the future of information and communication, in the form of quantum computing devices able to communicate and process massive amounts of data both efficiently and securely using quantum resources. Tremendous progress is continuously being made both technologically and theoretically in the pursuit of this long-term vision.

A primary goal of current quantum computing research is to find real-world applications of the quantum computers that will become available in the coming years. In order to arrive at these first applications, simultaneous progress on both hardware and algorithms is required. On the one hand, quantum hardware is making considerable advances. Small quantum computers capable of running representative algorithms were first made available in research laboratories, utilizing both trapped ion [1, 2] and superconducting qubits [3, 4]. Performance comparisons among different quantum computer hardware have been made for running a host of quantum computing tasks [5, 6]. A recent report on achieving quantum supremacy, where a task was performed by a quantum computer that cannot be simulated with any available classical computer [7], is an indication that powerful quantum computers will likely be available to researchers in the near future.

At the same time, considerable algorithmic work is underway in order to reduce the resources needed for implementing impactful quantum algorithms and bring them closer to the noisy-term intermediate scale quantum (NISQ) era. For example, more NISQ variants of amplitude estimation algorithms, a fundamental quantum procedure that is used in a large number of quantum algorithms, appeared recently [8, 9, 10, 11, 12].

In this work, we focus on quantum machine learning (QML). There are a number of reasons why machine learning is a good area to find applications of quantum computers. First, classical machine learning has proved to be an extremely powerful tool for a plethora of sectors, so enhancements from quantum computing will have a major impact. Second, we already know that fault-tolerant quantum computers with fast quantum access to classical data can provably offer advantages for many different applications

This project pursues a combined approach of quantum hardware and software that advances the state-of-the-art of QML implementations, bringing potential applications closer to reality. Even though the scale of the implementation remains a proof of concept, this project makes significant progress towards unblocking a number of theoretical and practical bottlenecks. In particular, we look at classification, one of the canonical problems in supervised learning with a vast number of applications. In classification, one uses a labelled dataset. For classification, and considering the early stage of the quantum computer technology, it was decided to do a straightforward comparison of a standard k-means clustering implementation from the python Sci-Kit Learn library against the standard implementation of QC Ware's Q-Means algorithm. Additionally, while classical applications can already take advantage of mature multi-processor

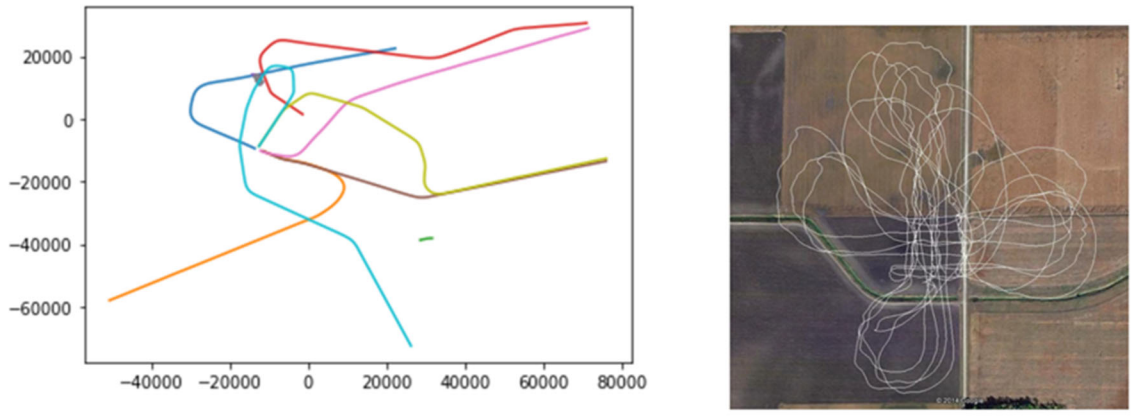
implementations, this is not yet the case for quantum. Thus, it was decided to compare the performance of the clustering algorithms on a single CPU against a single QPU. This should establish an adequate baseline from which to extrapolate, albeit with some caveats, the future performance of the quantum versus classical forms of the clustering algorithm. There are various choices for quantum hardware to run the experiment on, and we chose to use the IonQ quantum computer available via Amazon Web Services Braket platform for reasons addressed in this report.

3. METHODS, ASSUMPTIONS AND PROCEDURES

3.1. USE CASE ANALYSIS

The project began with an assessment of suitable use case data. A review of the machine learning clustering algorithm covered a number of possible applications in flight traffic management, air space command and control, and cybersecurity. Similarly, data sets were available online for certain specific applications in the form of data from application studies and in data sets curated for generic algorithmic research and competitions. However, for the quantum clustering algorithm Q-Means, it is known that the advantage over classical algorithms arises when considering input data with high dimensionality – that is, with a large number of features in each input data vector.

For that reason, we were attracted to flight path data where simple position (*x, y, z*) data versus time could be collected or simulated. When considering existing data sets, we looked at approach and landing flight paths into San Francisco International Airport [13] and flight paths used in agricultural remote sensing [14], as shown in Figure 1. The first example in particular explored principal component analysis followed by density-based clustering algorithms to classify nominal versus off-nominal flight paths for commercial air traffic. Rather than deciding on one or the other operational scenarios, we decided to consider a scenario of mixed operational activity that would represent areas where commercial, civilian, and military operations are ongoing, which is quite common in current USAF deployments.



(a) Representative approach flight paths at San Francisco International Airport.

(b) UAV flight path for agricultural remote sensing.

Figure 1. There is a large variety of flight paths in operational use, including (a) large-scale commercial flights approaching airports and (b) small-scale pattern-shaped flight path used in remote sensing.

3.1.1. Flight Path Types

In discussion with the AFRL customer, we settled on a set of flight path types as depicted in Figure 2.

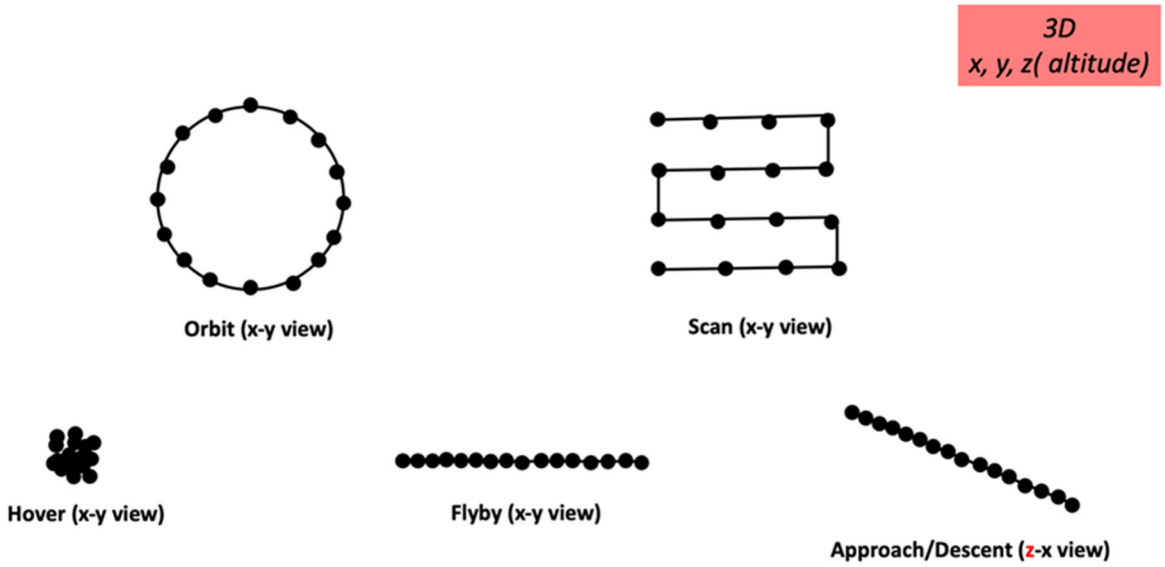


Figure 2. Various types of flight paths for different operational missions used in the Q-Means study.

The orbit type represents a station keeping air vehicle, perhaps engaged in surveying a fixed point or localized area. Examples would include Pegasus or Predator UAVs. The scan type represents an air vehicle flying a specified pattern, such as those used in mapping and remote sensing. The hover type represents a station-keeping air vehicle, such as a helicopter or drone. The flyby type represents probably the most common type, simply an aircraft flying through the area of interest, such as commercial airliners but perhaps also military aircraft *en route*. The approach type represent aircraft flying in a straight line, similar to the flyby type, but descending as they approach a landing at an airport.

An interesting feature of this selection of flight path types, aside from their plausible realism, is that they overlap in the characteristic speed and altitude regimes in which they operate. This is useful in that it is not too easy to discriminate flight path types based on a single feature and will occasionally lead to classification ambiguity for the synthetic data generated, as it would in real life operational situations. In other words, it won't give input data that is too easy to classify. This will allow testing of Q-Mean and k-means where they will have to make classifications on data points that are sometimes ambiguous and uncertain. Figure 3 shows the typical operational ranges of speed versus altitude for the various types used in generating synthetic data. The range in values were extracted from online searches of various air vehicles operating in the five mission categories. For example, the Flyby range of values is representative in speed and altitudes for various commercial airliners, commuter planes, and high-speed military aircraft. The Approach range of values is representative of airspeed versus altitude for the aircraft described in Flyby. The Scan range of values is representative of research grade aircraft, such as NASA's ER-2 High-Altitude

Airborne Science Aircraft, and smaller aircraft used in agricultural remote sensing. The Orbit range of values is representative of high-performance unmanned vehicles such as the Predator down to a lower-performance range for many smaller vehicles. And, finally, the Hover range of values is representative of helicopters and unmanned stationary drones.

3.1.2. Generating Synthetic Flight Path Types

A flight path model python script was developed that generated simulated flight path data for the various according to user specified inputs. The user can input how many of each type of flight path is to be generated. And the user can specify how many time steps of flight path data are generated for each path. As an example, if the user specifies n inputs=10, the model will generate 10 paths of each of the five types, or 50 flight paths in all. If the user specified n datapoints=10, the model would generate 10 triplets of (*x, y, z*) data, or 30 data points for each path. The scale of the model is set to be the airspace over a 200-mile square surface, extending up to an altitude of 100,000 feet. Currently, the model will generate each flight path to represent about 10 minutes of flight. An additional and very important input parameter is called variance. This is simply a coefficient to randomly generate variation form the nominal flight path speed and velocity. For example, setting the variance parameter equal to 0.0, will generate each flight path of a given type, with the exact same speed and horizon values, and thus will be easy for the clustering algorithms to classify them exactly without error. A variance parameter value of 1.0 will spread a 2σ variation of speed and velocity over the range given for each type in Figure 3. This will begin to allow for some ambiguity in the classification for some of the flight paths. Thus, the user can generate easy or hard data sets for classification.

3.1.3. Derived Features

An important part of any machine learning clustering algorithm is determining the right feature data to use. (*x, y, z*) data is problematic, and flight paths of the same type but space far apart in physical space, will not cluster together well. Guided by intuition, the authors felt that using (*speed, altitude, delta azimuth)* would be more suitable. The *delta azimuth* feature helps discriminate between straight flight paths and those with curvature or direction changes. This choice of feature data works well enough for the purposes of this study, but we can't say at this time that they are the best features to use. More investigation could be done on follow-on studies, and certainly, more discussion with operational experts would help clarify what the important things they're trying to capture in an operational airspace.

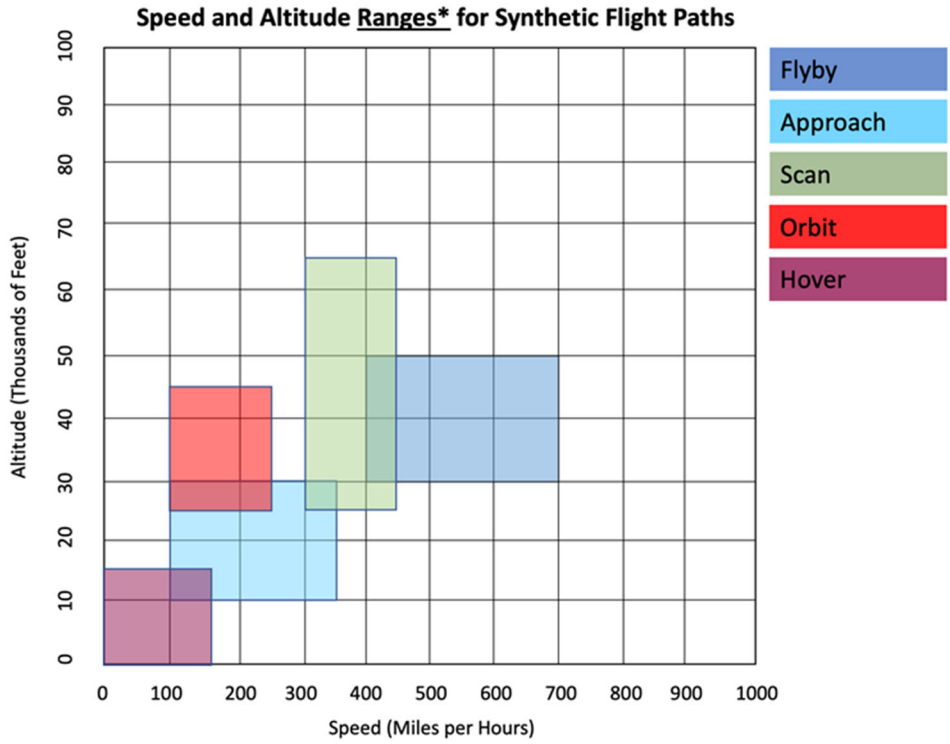
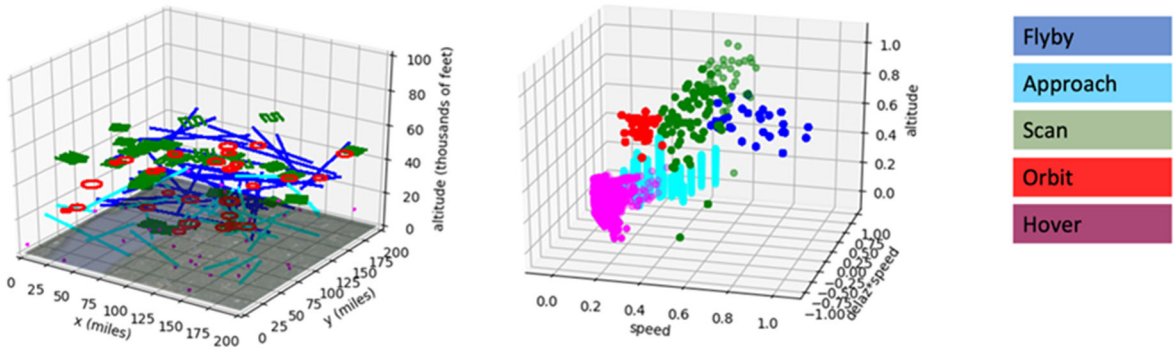


Figure 3. Altitude versus speed ranges for the various flight path types. (* The range of values were extracted from online searches of various air vehicles operating in the five mission categories.)

From this preliminary, intuitive approach, we've developed a script which inputs the flight path physical time series data vectors and outputs feature vectors consisting of normalized values of *speed*, *(delta azimuth)*speed*, and *altitude* for each time step. Figure 4 shows a representative set of synthetic flight path data and its derived feature set data.



Physical dimensions
30 instances * 5 types = 150 flight paths
50 time-coordinates * 3 (x,y,z) values =
150 spatial data points
Gives a [150, 150] input data array

Feature space
30 instances * 5 types = 150 flight paths
50 time-coordinates * 3 (speed, delta-azimuth, altitude) values =
150 spatial data points
Gives a [150, 150] input data array

Figure 4. A representative example of flight path physical data and their derived feature set vectors.

3.2. Q-Means Compared to K-Means

The Q-Means algorithm is analogous to the K-means algorithm. The outer control loop structure in both algorithms seek to define candidate cluster centers, then compute distances from input data points to candidate clusters, assign data points to the closest cluster, and then repeat the process with slightly different cluster centers, and iterating until the best fit is found. In K-means, the distances between all points and all cluster centers in each inner loop is computed exactly for all cases. For M data inputs of N features, and K cluster centers, this scales as MNK. However, a quantum algorithm has been invented that efficiently estimates the difference between data points with N features and the cluster centers. This quantum distance estimation scales as $M\sqrt{N}K$, giving it a quadratic improvement over the classical k-means algorithm. The Q-Means algorithm assembles several modules together to efficiently perform the quantum clustering. These include most prominently a data loader and a distance estimator [15].

3.3. Data loaders and distance estimation

We start by describing our data loaders [16]. Loading classical data as quantum states that can be efficiently used for further computation is an important step for QML applications, since most algorithms require efficient quantum access to data which is, and will likely remain, predominantly classical. A data loader is a procedure that, given access to a classical data point $x = (x_1, x_2, \ldots, x_d) \in R^d$, outputs a parametrized quantum circuit that prepares quantum states of the form

$$\frac{1}{\|x\|}\sum_{i=1}^{d} x_i |i\rangle \, .$$

Here, $|i\rangle$ is some representation of the numbers 1 through d. These states are called "amplitude encodings". They are not the only possible way to represent classical data as quantum states but

are by far the most interesting in terms of the quantum algorithms that can be applied to them. For example, they are the states one needs in order to start the quantum linear system solver procedure. To calculate the efficiency of our algorithms, we assume that quantum computers will have the ability to perform gates on different qubits in parallel. This is possible to achieve in most technologies for quantum hardware, including ion traps [17]. Section 3.4 discusses previous proposals for access to classical data.

The loader we will use for our experiment is a "parallel" unary loader that loads a data point with d features, each of which can be a real number, with exactly d qubits, $d - 1$ parametrized 2-qubit gates, and depth $\log d$. The parallel loader can be viewed as a part of a more extensive family of loaders with Q qubits and depth D, with $QD = O(d \log d)$. In particular, one can define an optimized loader with $2\sqrt{d}$ qubits and $\sqrt{d} \log d$ depth with $(d - 1)$ two- and three-qubit gates in total [16]. Note that the number of qubits we use, one per feature, is the same as in most quantum variational circuit proposals (e.g. [18, 19]).

Data loader construction The first step is a procedure that given access to a classical data point $x = (x_1, x_2, \ldots, x_d) \in \mathrm{R}^d$, pre-processes the classical data efficiently, i.e. spending only $O(d)$ total time, in order to create a set of parameters $\theta = (\theta_1, \theta_2, \ldots, \theta_{d-1}) \in \mathrm{R}^{d-1}$, that will be the parameters of the $(d - 1)$ two-qubit gates we will use in our quantum circuit. In the preprocessing, we also keep track of the norms of the vectors.

It is important here to notice that the classical memory is accessed once (we use read-once access to x) and the parameters θ are "stored inside the quantum circuit" (as the parameters of the quantum gates), which means that if we need to perform many operations with the specific data point (which is the case here and, for example, in training neural networks), we do not need to access the classical memory again, we just need to re-run the quantum circuit that already has the parameters in place.

Once we find the parameters that we need for our parametrized quantum circuit, we can define the architecture of our quantum circuit. For convenience in this exposition, we assume that d is a power of 2.

We will use a gate that has appeared with small variants with different names as partial SWAP, or fSIM, or Reconfigurable Beam Splitter, etc. We call this two-qubit parametrized gate RBS(θ) and we define it as

$$RBS(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\theta) & sin\,(\theta) & 0 \\ 0 & -sin\,(\theta) & cos\,(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{1}$$

We can think of this gate as a simple rotation by an angle θ on the two-dimensional subspace spanned by the vectors $\{|10\rangle , |01\rangle\}$ and an identity in the other subspace spanned by $\{|00\rangle , |11\rangle\}$. We denote by RBS†($\theta$) the adjoint gate for which we have RBS†(θ) = RBS($-\theta$).

We can now describe the circuit itself. We start by putting the first qubit in state $|1\rangle$, while the remaining $d-1$ qubits remain in state $|0\rangle$. Then, we use the first parameter θ_1 in an RBS gate in order to "split" this '1' between the first and the $d/2$-th qubit. Then, we use the next two parameters θ_2, θ_3 for the next layer of two RBS gates, where again in superposition we "split" the '1' into the four qubits with indices (1, $d/4$, $2d/4$, $3d/4$) and this continues for exactly log d layers until at the end of the circuit we have created exactly the state

$$|x\rangle = \frac{1}{\|x\|}\sum_{i=1}^{d} x_i|e_i\rangle \tag{2}$$

where the states $|e_i\rangle$ are unary representations of the numbers 1 to d, using d qubits. The circuit appears in **Error! Reference source not found.**.

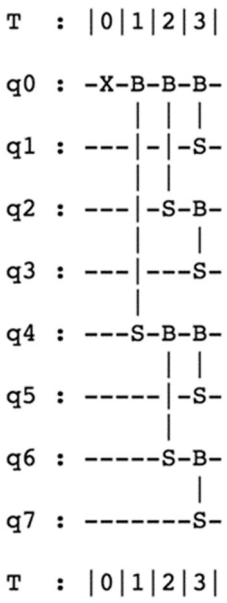


Figure 5. The data loader circuit for an 8-dimensional data point. The angles of the *RBS*(θ) gates starting from left to right and top to bot- tom correspond to (θ_1, θ_2, ..., θ_7).

Let us make some remarks about the data loader circuits. First, we will see in the following sections that the circuits are quite robust to noise and amenable to efficient error mitigation techniques. Second, we use a single type of two-qubit gate that is native or quasi-native to different hardware platforms. Third, we note that the connectivity of the circuit is quite local, where most of the qubits interact with very few qubits (for example 7/8 of the qubits need at most 4 neighboring connections) while the maximum number of interacting neighbors for any qubit is log d. Here, we could take advantage of the full connectivity of trapped ion quantum processors, such as the IonQ hardware platform, so we can apply all gates directly. On a grid architecture, such as on superconducting flux qubits, one would need to embed the circuit on the grid which asymptotically requires no more than doubling the number of qubits.

Distance estimation circuit The power of the data loaders comes from the operations that one can do once the data is loaded into such amplitude encoding quantum states. In this work, we use the data loader circuits to perform a fundamental operation at the core of supervised and unsupervised similarity-based learning, which is the estimation of the distance between data points.

Given two vectors x and y corresponding to classical data points, the Euclidean distance, namely $l_{xy} = \|x - y\|$ [16] is given by

$$l_{xy} = \sqrt{\|x\|^2 + \|y\|^2 + 2\|x\|\|y\|c_{xy}} \tag{3}$$

where $c_{xy} = \langle x|y\rangle$ is the inner product of the two normalized vectors. Here we describe a circuit to estimate c_{xy} which is combined with the classically calculated vector norms to obtain l_{xy}, as depicted in **Error! Reference source not found.**.

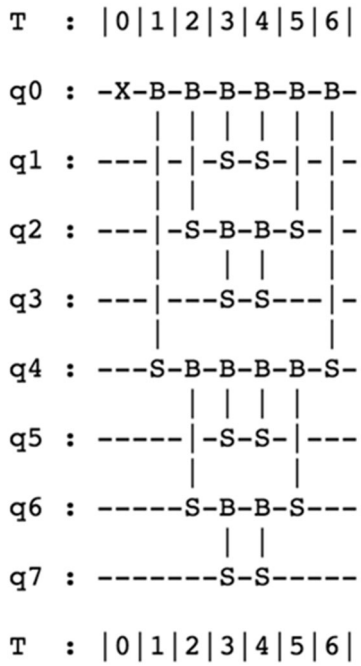


Figure 6. The distance estimation circuit for two 8-dimensional data points. The circuit in time steps 0 − 3 corresponds to the parallel loader for the first data point and the circuit in time steps 4 − 6 corresponds to the inverse parallel loader circuit for the second data point (excluding the last X gate). The probability the first qubit is measured in state $|1\rangle$ is exactly the square of the inner product of the two data points.

In fact, here we will only discuss one of the variants of the distance estimation circuits which works for the case where the inner product between the data points is positive, which is usually the case for image classification where the data points have all non-negative coordinates. It is not hard to extend the circuit with one extra qubit to deal with the case of also non-positive inner products.

The distance estimation circuit for two data points of dimension d uses d qubits, $2(d-1)$ two-qubit parametrized gates, 2 log d depth, and allows us to measure at the end of the circuit a qubit whose probability of giving the outcome $|1\rangle$ is exactly the square of the inner product between the two normalized data points. From this, one can easily estimate the inner product and the distance between the original data points. The distance estimation circuit is explained in Figure 6.

We can also notice a simplification we can make in the circuit that will reduce the number of gates and depth. This reduces the number of gates of the circuit to $3d/2 - 2$ and reduces the depth by one. This is shown in **Error! Reference source not found.**.

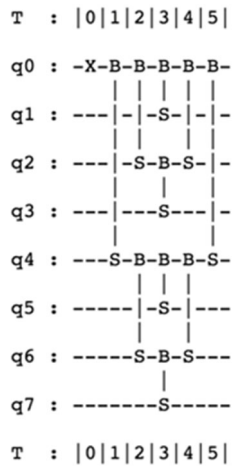


Figure 7. The optimized distance estimation circuit for two 8-dimensional data points. For each pair of RBS gates that are applied to the same consecutive qubits in time steps 3 and 4 in the circuit are combined to one gate whose parameter θ is equal to $\theta_1 + \theta_2$, where θ_1 is the parameter of the first gate and θ_2 is the parameter of the second gate.

3.4. Previous Proposals for Loading Quantum Data into Quantum States

There have been several proposals for acquiring fast quantum access to classical data that loosely go under the name of QRAM (Quantum Random Access Memory). A QRAM, as described in [20, 21], in some sense would be a specific hardware device that could "natively" access classical data in superposition, thus having the ability to create quantum states like the one defined above in logarithmic time. Given the fact that such specialized hardware devices do not yet exist, nor do they seem to be easy to implement, there have been proposals for using quantum circuits to perform similar operations. For example, a circuit to perform the bucket brigade architecture was defined in [22], where a circuit with $O(d)$ qubits and $O(d)$ depth was described and also proven to be robust up to a level of noise. A more "brute force" way of loading a d-dimensional classical data point is through a multiplexer-type circuit, where one can use only $O(logd)$ qubits but for each data point one needs to sequentially apply d log d-qubit-controlled gates, which makes it quite impractical. Another direction is loading classical data using a unary encoding. This was used in [23] to describe finance applications, where the circuit used $O(d)$ qubits and had $O(d)$ depth. A parallel circuit for specifically creating the W state also appeared in [24].

3.5. Method for Finding the Gate Parameters

We describe here the procedure that given access to a classical data point $x = (x_1, x_2, ..., x_d) \in \mathrm{R}^d$, preprocesses the classical data efficiently – i.e., spending only $O(d)$ total time, in order to create a set of parameters $\theta = (\theta_1, \theta_2, \ldots, \theta_{d-1}) \in \mathrm{R}^{d-1}$, that will be the parameters of the $(d-1)$ two-qubit gates we will use in our quantum circuit.

The data structure for storing θ is in fact the one used in [25] but note that there we assumed that we have quantum access to these parameters (in the sense of being able to query these parameters in superposition) while here we will compute and store these parameters classically and also encode them as the parameters of the gates used in the quantum circuit.

At a high level, we think of the coordinates x_i as the leaves of a binary tree of depth log d. The parameters θ correspond to the values of the internal tree nodes, starting from the root and going towards the leaves.

We first consider the parameter series $(r_1, r_2, ..., r_{d-1})$. For the last $d/2$ values $(r_{d/2}, ..., r_{d-1})$, we define an index j that takes values in the interval $[1, d/2]$ and define the values as

$$r_{\frac{d}{2}+j-1} = \sqrt{x_{2j}^2 + x_{2j-1}^2}$$

For the first $d/2 - 1$ values, namely the values of $(r_1, r_2, \ldots, r_{d/2-1})$, and for j in $[1, d/2]$, we define

$$r_j = \sqrt{r_{2j+1}^2 + r_{2j}^2}$$

We can now define the set of angles $\theta = (\theta_1, \theta_2, ..., \theta_{d-1})$ in the following way. We start by defining the last $d/2$ values $(\theta_{d/2}, \ldots, \theta_{d-1})$. To do so, we define an index j that takes values in the interval $[1, d/2]$ and define the values as

$$\theta_{\frac{d}{2}+j-1} = arccos\left(\frac{x_{2j-1}}{r_{\frac{d}{2}+j-1}}\right), \text{ if } x_{2j} \text{ is positive,}$$

$$\theta_{\frac{d}{2}+j-1} = 2\pi arccos\left(\frac{x_{2j-1}}{r_{\frac{d}{2}+j-1}}\right), \text{ if } x_{2j} \text{ is negative.}$$

For the first $d/2 - 1$ values, namely the values for $j \in [1, d/2]$, we define

$$\theta_j = arccos\left(\frac{r_{2j}}{r_j}\right).$$

Note that we can easily perform these calculations in a read-once way, where for every x_i we update the values that are on the path from the i-th leaf to the root. This also implies that when one

coordinate of the data is updated, then the time to update the θ parameters is only logarithmic, since only log d values of r and of θ need to be updated.

3.6. Method for an Optimized Data Loader

An interesting extension of the parallel data loader we defined above is that we can trade off qubits with depth and keep the number of overall gates $d-1$ (in this case, both RBS and controlled-RBS gates).

For example, we can use $2\sqrt{d}$ qubits and $O\left(\sqrt{d}logd\right)$ depth [16]. The circuit is quite simple. If one thinks of the d-dimensional vector as a $\sqrt{d} \times \sqrt{d}$ matrix, then we can index the coordinates of the vector using two registers (one each for the row and column) and create the state

$$|x\rangle = \frac{1}{\|x\|}\sum_{i,j=1}^{\sqrt{d}} x_{ij}|e_i\rangle \left|e_j\right\rangle .$$

For this circuit, we find, in the same way as for the parallel loader, the values θ and then create the following circuit in **Error! Reference source not found.**. We start with a parallel loader for a $\sqrt{d}$-dimensional vector using the first $\sqrt{d}$ angles θ (which corresponds to a vector of the norms of the rows of the matrix) and then, controlled on each of the $\sqrt{d}$ qubits we perform a controlled parallel loader corresponding to each row of the matrix.

Notice that naively the depth of the circuit is$O(dlogd)$, but it is easy to see that one can interleave the gates of the controlled-parallel loaders to an overall depth of $O\left(\sqrt{d}logd\right)$. We will not use this circuit in this study, but such circuits can be useful both for loading vectors and in particular matrices for linear algebraic computations.

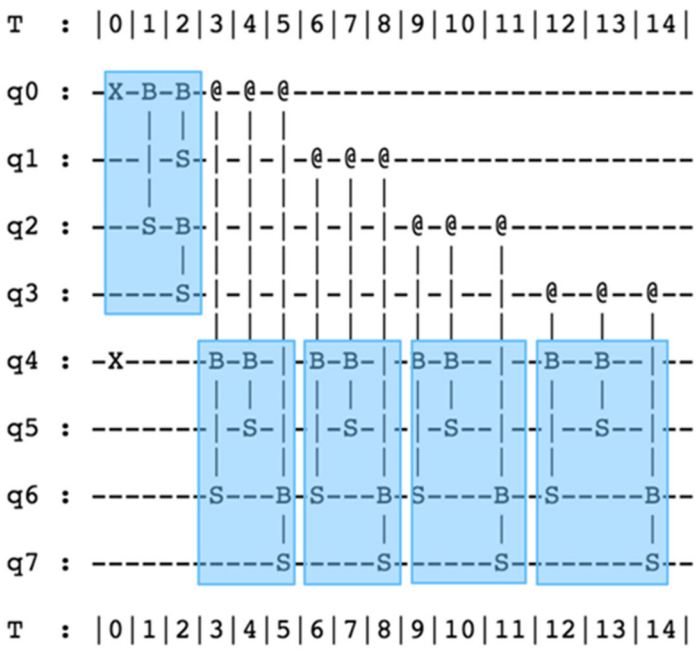


Figure 8. An optimized loader for a 16-dimensional data point, seen as a 4 × 4 matrix. The blue boxes correspond to the parallel loader from Figure 7 and its controlled versions.

We show an example of how to compute quantum processor resources for data loading as shown in **Error! Reference source not found.**. Using the gray, dashed line in the plots above to create a hypothetical example: Suppose that the current state-of-the-art hardware is a 50-qubit quantum computer (see the grey dashed line on the right plot) with the ability to execute a circuit with depth 100 with reasonably low error (see the grey dashed line on the left plot). Note that these are roughly the specifications of the Google Sycamore chip that was used to perform the first demonstration of quantum supremacy. In that sense, this is a reasonably accurate picture of the state-of-the-art in superconducting hardware.

With such hardware, the maximum dimensionality we can work with, if we were to use the sequential loader, is a dimensionality of about 2^5 to 32 (i.e., see the intersection of the grey dashed line and the blue solid line in the left plot). At this point, the circuit depth required is about 100, which is our state-of-the-art assumption on the maximum executable circuit depth.

Similarly, the maximum dimensionality we can work with if we were to use the parallel loader is also a dimensionality of about $2^5 = 32$ (i.e., see the intersection of the grey dashed line and the orange solid line in the right plot). At this point, the number of qubits required is about 50, which is our state-of- the-art assumption on the number of qubits.

Finally, using the tailored loader, the maximum dimensionality we can work with is about $2^9 = 512$ (i.e., see the intersection of grey dashed line and green solid line in the left plot above). At this point the circuit depth required is about 100, which our state-of-the-art assumption on the maximum executable circuit depth. In short, using our tailored loader, we are able to increase the

maximum dimensionality of data that we can work with on state-of-the-art hardware from about 2^5 to about 2^9.

Results will vary depending on the hardware under consideration. However, the general idea still stands. Using the tailored loader, we can trade off qubit quality for number of qubits, and vice versa. By choosing the trade-off appropriately, as dictated by the hardware's specifications, we are able to make more efficient use of the hardware.

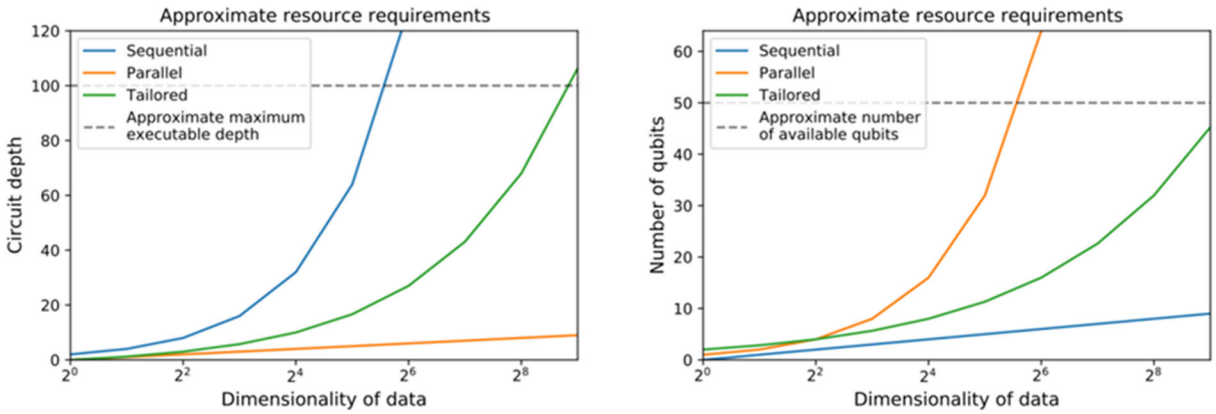


Figure 9. Shows the trade-off between number of qubits and circuit depth to achieve the optimal data loading circuit.

3.7. Method for Estimating the Distance

The distance estimation circuit is explained in **Error! Reference source not found.** above. Here we see how from this circuit we can estimate the square of the inner product between the two vectors. After the first part of the circuit, namely the loader for the vector x (time steps 0-3), the state of the circuit is $|x\rangle$, as in (2). One can rewrite this state in the basis $\{|y\rangle, |y^{\perp}\rangle\}$ as

$$\langle x, y\rangle |y\rangle + \sqrt{1 - |\langle x, y\rangle|^2} |y^{\perp}\rangle .$$

Once the state goes through the inverse loader circuit for y the first part of the superposition gets transformed into the state $|e_1\rangle$ (which would go to the state $|0\rangle$ after an X gate on the first qubit), and the second part of the superposition goes to a superposition of states $|e_j\rangle$ orthogonal to $|e_1\rangle$,

$$\langle x, y\rangle |e_1\rangle + \sqrt{1 - |\langle x, y\rangle|^2} |e_1{}^{\perp}\rangle .$$

It is now clear that after measuring the circuit (either all qubits or just the first qubit), the probability of getting $|1\rangle$ in the first qubit is exactly the square of the inner product of the two data points.

3.8. Runtime and scalability

The quantum advantage comes from the distance estimation procedure which is based on the data loader circuits. As described above, the circuit for estimating the distance of two d-dimensional

data points has depth $2logd$. Theoretically, for an estimation of the distance up to accuracy ε one needs to run the circuit $N_s = O(1/\epsilon^2)$ times. In the future, amplitude estimation can be combined with this circuit in order to reduce the overall time to $O(log\ d/\epsilon)$. The accuracy required depends on how well-classifiable our data set is, that is, whether most points are close to a centroid, or they are mostly distributed equidistantly from the centroids. This number does not really depend on the dimension of the data set and in all data sets we considered, an approximation to the distance up to 0.1 for most points and 0.03 for a few difficult to classify points suffices. We argue that the number of shots will increase slowly as the problem sizes scale up.

Since the cost of calculating the circuit parameters is a one-off cost, if we want to estimate the distance between k centroids and n data points all of dimension d, then the quantum circuits will need d qubits and the running time would be of the form $O(kd+nd+kn$ log(d)/ε). The first term corresponds to pre-processing the centroids, the second term to pre-processing the new data points and the third term to estimating the distances between each data point and each centroid and assigning a label. The basic classical Nearest Centroid algorithm takes time $O(nkd)$. One can design different classical Nearest Centroid algorithms that also sample using special data structures which will still be quadratically worse than a fully quantum one with respect to ε.

3.9. Caveats on Overall Run Time

While the Q-means algorithm has a scaling advantage in terms of computational steps, a strict comparison of total computation time for k-means and q-means would need to include the clock cycles of the classical and quantum processors and latency times in sharing data between processors, for example if the quantum processor was remote from the classical processor. However, this study effort intends to do a theoretical comparison at this stage of quantum computing hardware. As the quantum computing hardware matures to the point of providing advantage over classical in individual algorithms, the overall integration of the computational workflow would have to reduce latencies and optimize for run-time performance.

3.10. The quantum nearest centroid classifier

We now have all the necessary ingredients to implement the quantum Nearest Centroid classification circuit. As we have said, this is but a first, simple application of the above tools which can readily be used for other machine learning applications such as nearest neighbor classifiers or k-means clustering for unsupervised learning, where neural network techniques are not available. Let us start by briefly defining the Nearest Centroid algorithm in the classical setting. The first part of the algorithm is to use the training data to fit the model. This is a very simple operation of finding the average point of each class of data, meaning one adds all points with the same label and finds the centroid of each class. This part will be done classically, and one can think of this cost as a one-time offline cost.

In the quantum case, one will still find the centroids classically and then pre-process each to find its norm and parameters for the gates of the data loader circuits. This does not change the asymptotic time of this step.

We now look at the second part of the Nearest Centroid algorithm which is the predict phase. Here, we want to assign a label to a number of test data points and for that we first estimate the distance between each data point and each centroid. For each data point we then assign the label of the centroid which is nearest to it.

The quantum Nearest Centroid is rather straightforward, it follows the steps of the classical k-means algorithm apart from the fact that whenever one needs to estimate the distance between a data point and a centroid, we do this using the distance estimator circuit defined above.

3.11. QUANTUM COMPUTER CLASSICAL SIMULATORS

3.11.1. quasar

QC Ware's 'quasar' simulator was developed by QC Ware to provide the flexibility to add the necessary features we desired in a simulator and to make it available to our customers. It is a full simulation of a quantum computing unitary matrix multiplication. There are of course a variety of quantum simulators offered by other companies and academia. Of special note are quantum simulators offered by the quantum computing hardware vendors, as these offer features especially relevant to their available hardware. QC Ware maintains translators for our quasar-implemented circuits to the vendor-supplied circuit APIs. 'quasar' allows the full simulation of quantum circuits of up to 30 qubits on a CPU and on a GPU when speedier performance is desired.

3.11.2. blazar

'blazar' is QC Ware's simulator designed especially for the RBS gates used in the data loader and distance estimation algorithms. It was noted that the structure of the quantum distance estimation allows for a more efficient classical simulation, enabling shortcuts to be made rather than using the full simulation offered by quasar and other quantum simulators. The structure of the distance estimation circuit is restricted to a much smaller subspace of the circuit. Thus, the classical computations can be tailored to evaluate only this subspace and not the full 2^N Hilbert space of a general quantum circuit. The user is not restricted to which gates can be used to compose the RBS gates, but rather it is the implementation of their structure in the circuit that properly defines the subspace and allows blazar to invoke the short cut in computing the results in the subspace. This enables larger problem sizes of input data to be analyzed classically by the Q-Means algorithm. This will be helpful in comparing performance of quantum simulated Q-Means solutions against classical k-means solutions.

3.11.3. Suitability of flight path use case

The flight path model code is suitable for generating a variety of flight path types (classifications) of user defined speed, altitude, and variations of these within a flight path and between different flight paths. The best features to use at this point is not yet determined and is not the primary purpose of the study at this point. However, we chose a set of features derived from the physical parameters of the flight path that were guided by intuition to help discriminate between different

flight paths. However, variety of features one could chose is only limited by imagination and then confirmed by testing. Additional work could be done in follow-on work to consider different feature sets, however this would best include subject matter practitioners who could insert more realism into what the operational needs are.

3.11.4. Implementation of Q-Means Algorithm

The data loader works well, enabling large problems to be fit into the circuit. The q-means algorithm's implementation of the distance estimation routine works well, accurately classifying flight paths with similar accuracy as k-means in the trial runs we've done so far on quantum computing simulators. The full simulation on quasar was successful but only for very small problem sizes, for example 15 flight paths by 4 features. Testing on the newly developed blazar, which takes advantage of structure of the distance estimation computation, enables much larger problem sizes to be simulated.

4. RESULTS AND DISCUSSION

4.1. BENCHMARKING CURRENT CLASSICAL APPROACHES

4.1.1. k-Means Algorithm

The quantum Q-Means algorithm is analogous to the classical k-Means algorithm. The outer control loop structure in both algorithms seeks to define candidate cluster centers, then compute distances from input data points to candidate clusters, assign data points to the closest cluster, and then repeat the process with slightly different cluster centers, and iterating until the best fit is found. In k-Means, the distances between all points and all cluster centers in each inner loop is computed exactly for all cases (where Q-Means uses a probabilistic quantum algorithm for these distance measurements). For N data inputs of d features, and k cluster centers, this scales as $O(Ndk)$, ignoring for now the ambiguity of how many iterations would be needed to converge to acceptable stopping conditions.

4.1.2. Performance Testing Methodology

We begin by using the selected (in this case, simulated) data, which included its reformulation into a derived dataset more suitable for the clustering algorithm in terms of discriminating features. While classical k-means scales linearly in number of clusters, k, the number of data inputs, N, the number of data features, d, per each iteration, the overall performance also depends on the number of iterations needed for convergence, which is not strictly linear and might vary with problem type and instance.

We used Sci-Kit Learn python libraries (k-means++) to evaluate the scaling performance of k-Means on the synthetically generated flight path data from this study. The Sci-Kit Learn libraries already implement by default the k-means++ algorithm for its k-means implementation. The most significant improvement in the k-means++ version is that starting points for cluster centers are chosen in a more intelligent manner, avoiding the poor results or longer running times that can occur when a pathological edge case set of cluster starting points are chosen completely randomly. The q-means algorithm was designed following the implementation of the Sci-Kit Learn k-means, often using some of the same open-source code, so both algorithms use the k-means++ cluster starting point algorithms, which tries to find better starting locations for the centroids.

Synthetic flight path data was regenerated for each timing run, for 10 different timing runs at each problem size, to account for random problem instances being easier or harder by chance, and then averaged over the 10 runs to arrive at the benchmark runtime. We set k=5 for the five types of flight paths.

We also generated synthetic clusters of Gaussian-distributed random points, typical for generic testing of clustering algorithms, to assess whether the k-Means behavior for the flight path data was typical or not compared to generic Gaussian clusters.

4.1.3. Classical Benchmarking Results

We explored a variety of settings of problem sizes that were both comparable to the possible problem sizes that could fit on quantum hardware in the near- to mid-future and also convenient to run numerous test sizes classically. We then executed a range of problem sizes for both synthetic flight path data and Gaussian clusters with varying number of data points, *N*, and feature dimensionality, *d*. For completeness, the run time data for these various runs is shown in **Error! Reference source not found.**.

Table 1. Run time data for different cases spanning a range of number of datapoints and a range of dimensionality of each data point. There are two classes of data: the top rows are for flight path data, and the bottom rows are for Gaussian clusters.

K-Means Run Times (CPU*sec)									
Flight Paths	Features per Row								
Data Rows	500	+/-	1000	+/-	1500	+/-	2000	+/-	
500	0.222	0.045	0.256	0.046	0.269	0.045	0.329	0.049	
1000	0.653	0.158	0.679	0.176	0.908	0.073	0.899	0.075	
1500	0.984	0.0311	1.065	0.028	1.127	0.042	1.165	0.022	
2000	1.168	0.023	1.262	0.056	1.315	0.032	1.382	0.033	
K-Means Run Times (CPU*sec)									
Gaussian Points	Features per Row								
Data Rows	500	+/-	1000	+/-	1500	+/-	2000	+/-	
500	0.22	0.047	0.25	0.047	0.278	0.063	0.321	0.04	
1000	0.607	0.226	0.768	0.154	0.916	0.064	1.004	0.044	
1500	1.019	0.044	1.102	0.021	1.173	0.067	1.201	0.033	
2000	1.192	0.022	1.281	0.032	1.354	0.042	1.422	0.035	

To interpret the table of data, we examined the scaling behavior of runtime versus problem size and number of features (dimensionality) separately. First let's look at scaling versus problem size, *N*, as shown in **Error! Reference source not found.**. These runs were made for several different settings of dimensionality, but the overall behavior is approximately linear as expected, with the slope of the linearity turning down a bit fo larger problem sizes, presumably due to decreasing number of iterations or tolerance for the stopping conditions once the cluster centers converge to a good enough final value.

Similarly, the scaling behavior of runtime versus dimensionality of features, *d*, is shown in Figure 11. And we see a similar nearly linear relation, as expected. However, comparing the two plots, we notice that at this scale of the problem size, the overall runtime is less dependent on dimensionality than it is on number of data points.

We see very similar behavior in both absolute magnitude and scaling of run time performance for the Gaussian clusters as we observe for the flight path data, as shown in Figure 12 and Figure 13 shown above. At this scale of the problem with the number of data points and number of features in each data point, we see the flight path clustering performs in family with a standard, well-behaved dataset such as the Gaussian clusters.

At the current problem sizes of number of data points times the number of features, *Nd*, the number of datapoints, N, is about one order of magnitude dominant in sensitivity, so overall speedup from Q-Means would need to consider more features to see if the behavior is different in the regime

where the quantum Q-Means would be anticipated to provide and advantage of the classical k-Means. But this performance baseline will be adequate to continue into the next phase of the project and begin preparations for testing on real quantum computing hardware.

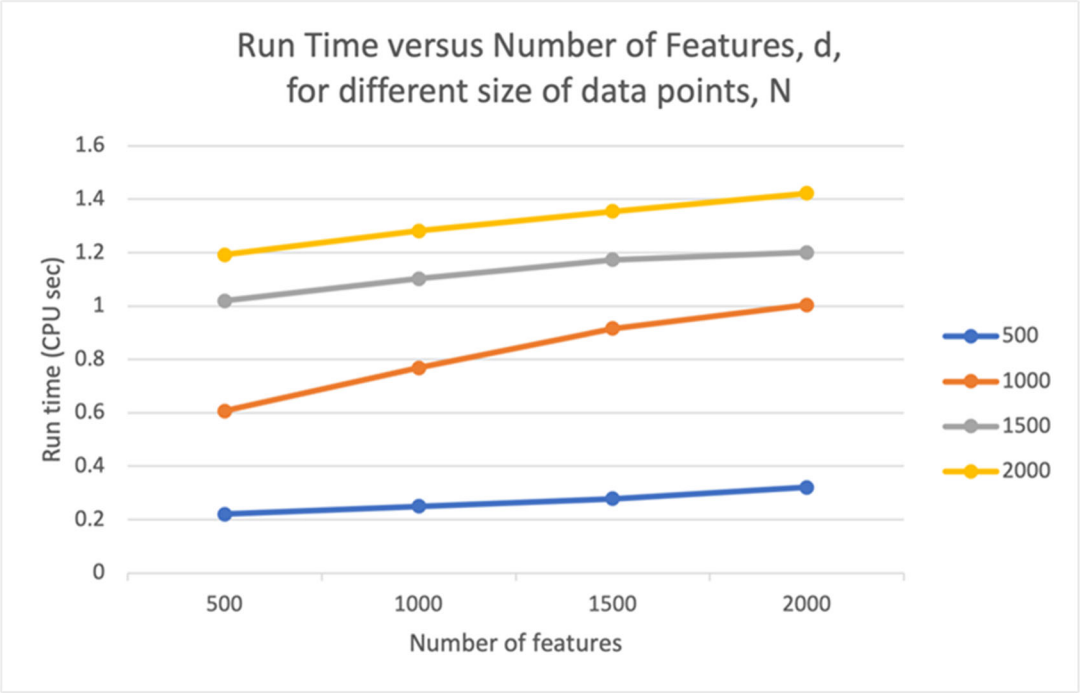


Figure 10. Scaling of runtime versus number of data points, *N*

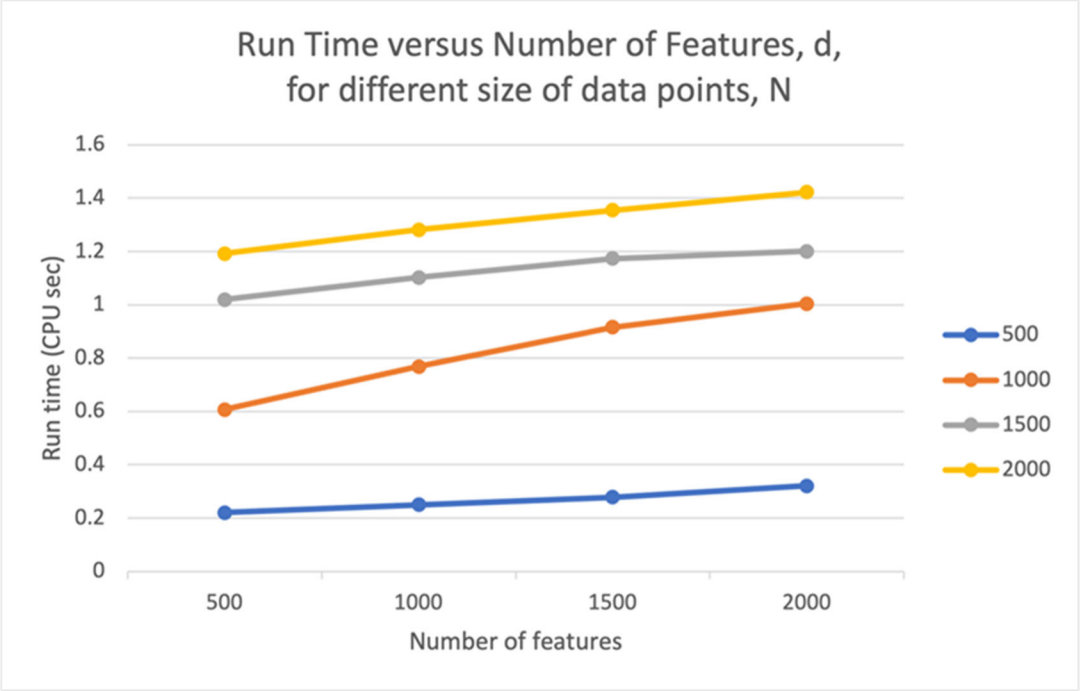


Figure 11. Scaling of runtime versus data point dimensionality, *d*.

Gaussian Cluster Data Run Time

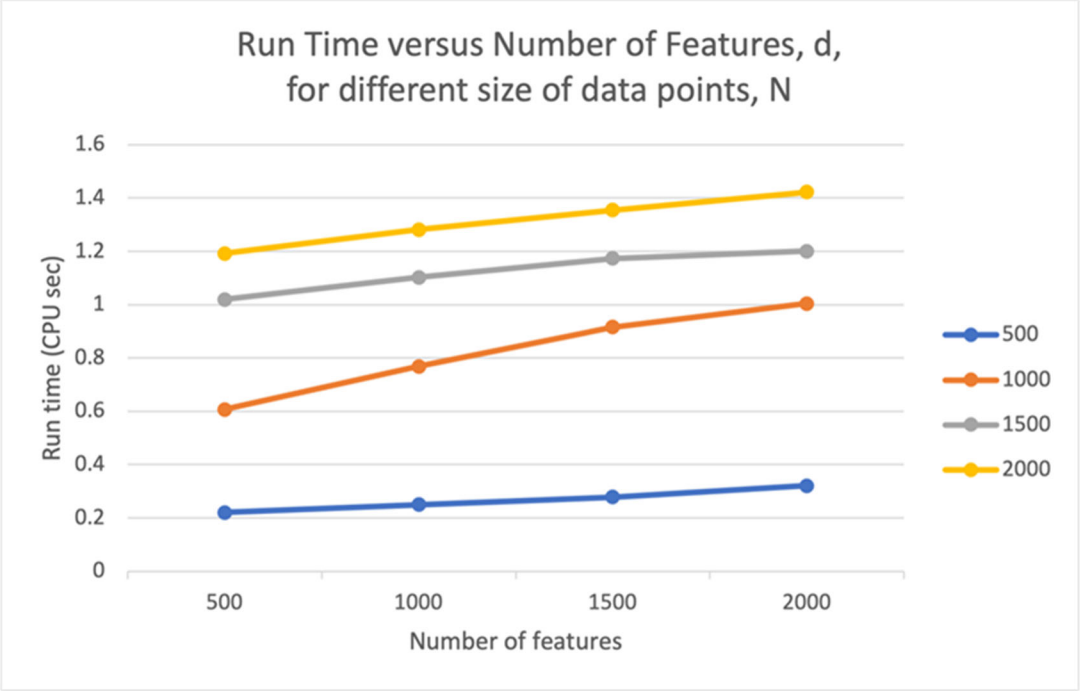


Figure 12. Scaling of runtime versus number of data points, *N*

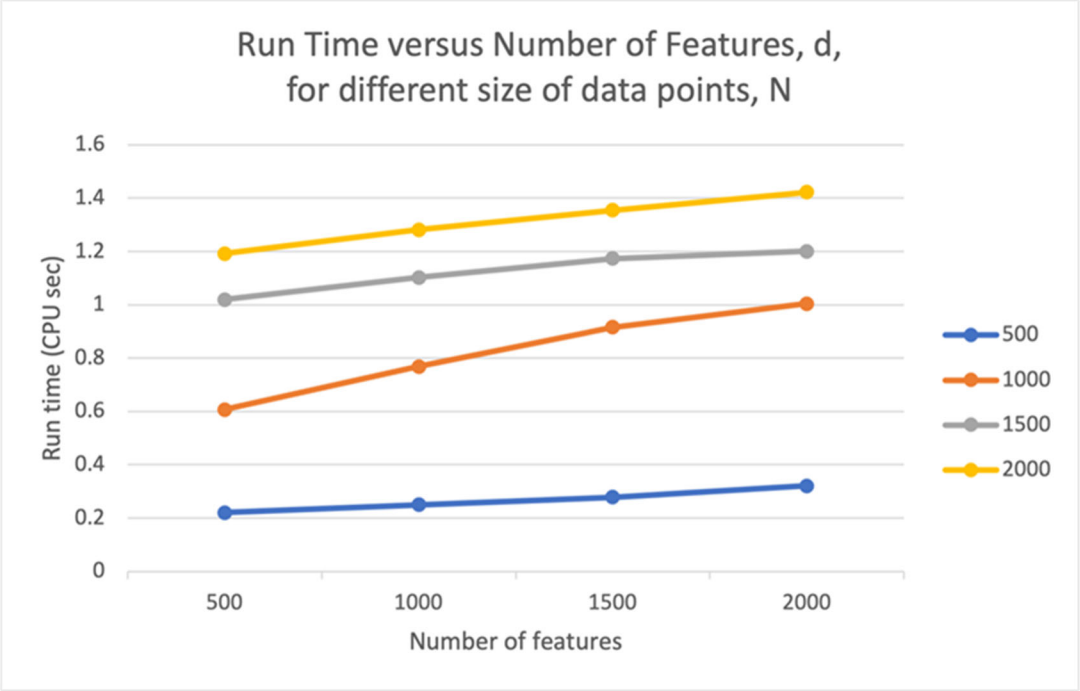


Figure 13. Scaling of runtime versus data point dimensionality, *d*.

4.2. ANALYZE QUANTUM COMPUTER RESOURCE USAGE

The next phase of the research was to determine the resources needed to run the q-means algorithm on existing quantum computer hardware. This includes both how large a problem can fit into the available qubits and how long the quantum circuit can run to reliably compute a result. In this case, we are limited in the choice of available quantum hardware and that all of them can still only run problems of small sizes.

Table 2. Qubits and circuit depth required to run flight path Q-Means problems of various problem sizes.

Problem Size	Parallel loader		Optimal loader	
time increments	Qubits	Depth	Qubits	Depth
5	9	5	8	21
50	144	9	32	145
500	1494	12	96	416
5000	14994	15	256	1921

First let us examine the size of the circuit needed to run the flight path clustering problem. The quantum circuit size is determined by how many data points in each flight path we include (not by the overall number of flight paths). As we include longer and longer flight paths, we will need more and more qubits in quantum circuits of greater depth to load into the quantum computer to measure the distance between each flight path and the candidate cluster centers. Table 2 shows the number of qubits and circuit depths for various problem sizes. Only a problem size of around five time increments of flight path data is feasible to run on today's quantum computing hardware. Problems of more interesting and realistic size (thousands of time increments) would require quantum computers of thousands of qubits and circuit depths of thousands or more.

Figure 14 shows an example flight path problem suitable for running in this project. Top-left shows the problem description in terms of number of data points (2) for each of the five flight path types (thus N=10 flight paths in total) and then number of data points (5) in the time series for each flight path. Bottom-left shows a picture of the physical flight paths generated. Bottom-right shows the derived data points. Top-right shows the size of circuits generated to load the data into a quantum computer, in this case requiring a minimum of 9 qubits, 16 gates and a circuit depth of 5.

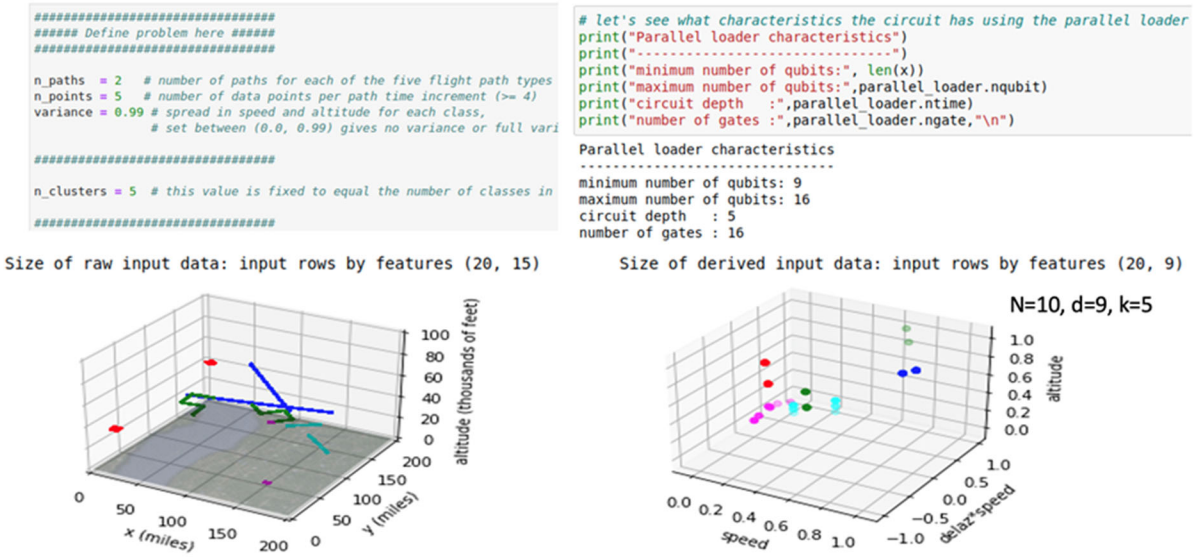


Figure 14. An example flight path problem suitable for running in this project.

4.3. OPTIMIZE ALGORITHM FOR SPECIFIC HARDWARE

4.3.1. Comparing IonQ, Rigetti, and IBM

The quantum computers of today contain less than 50 qubits. And we knew from our own experience and on the published results of the quantum computing community, that quantum circuits using less than 20 qubits are the most one can expect to execute a computation before noise limits the validity of results. Thus, we looked at the ability of various quantum computers commercially available to measure the distance between two vectors—this is the critical quantum

measurement made in the Q-Means algorithm. **Error! Reference source not found.** shows the results of an experiment in measuring distances between vectors for various quantum computing hardware. The Rigetti Computing and IonQ quantum computers are available via Amazon Web Services Braket and the two IBM quantum computers are available via IBM. All measurements were done using 100 shots or samples. In each of the four plots, the quantum measurements are shown in red against a background of simulated noiseless quantum measurements shown in blue. The better the red data points align with the blue data points, the more accurate the quantum measurement. The excursion of the measured red data points above the theoretically expected blue points is due to error in the quantum measurement, which systematically drives the measure value upward. The IonQ quantum computer shows the best performance for this distance measurement used in the Q-Means algorithm.

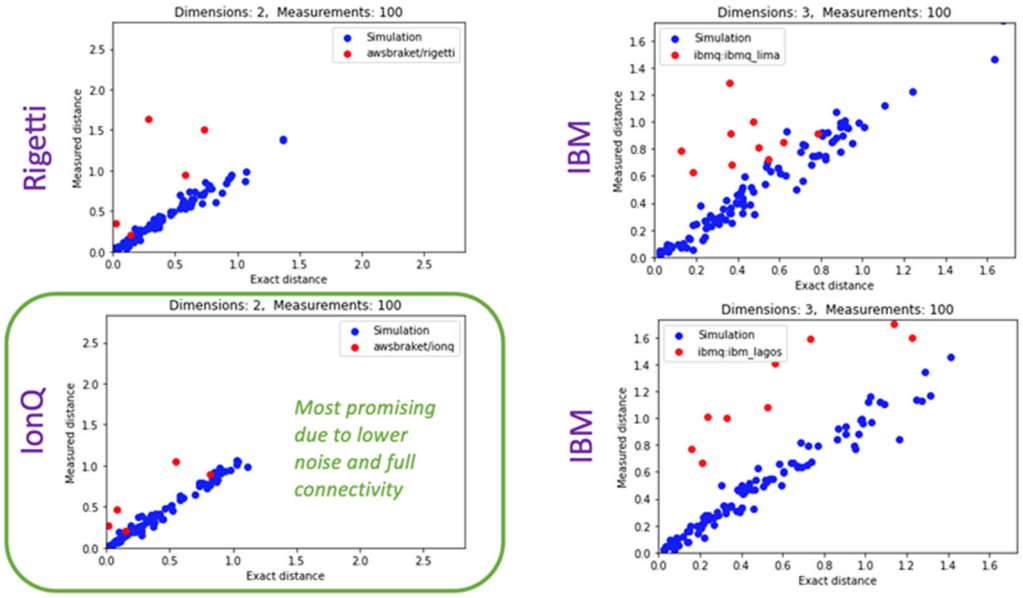


Figure 15. Measurement of random 2-dimensional vector pairs on various quantum computers.

4.3.2. Additional comparison of IonQ, Rigetti, and Lucy

A further study was done comparing the ability of quantum hardware in more detail as the hardware experiments were conducted. In particular, we examined the ability of various hardware to measure higher dimensional data, up to d=9 and focused on the quantum dot product measurement of two vectors, qdot, which is more fundamental than the distance measurement shown above in that the distance measurement is derived from additional classical calculations from the qdot measurement. We also varied the number of shots used to take each qdot measurement as this is important in determining the cost of running the problem on quantum hardware. For example, if we require 1000 shots to get a good qdot measurement versus 100 shots, those 10 times number of shots will mean that the overall problem will be 10 times more expensive to execute on quantum hardware.

Figure 16 shows the measurement of the quantum dot product of random 2-dimensional vector pairs on the IonQ quantum computer for different dimensions and different numbers of shots. The

top row shows measurements of 100 shots for different dimensions (d=3, 6, 9). The bottom row shows measurements of 1000 shots for the same dimensions. For the qdot measurement, the line with slope of 1.0 in the figure represents the theoretically correct value of quantumly measured versus classically exact value. Deviation from this center line is due partly to quantum randomness and partly to noise. More shots will decrease the randomness and spread in the data, while the noise lowers the value of the measurement and cannot be corrected.

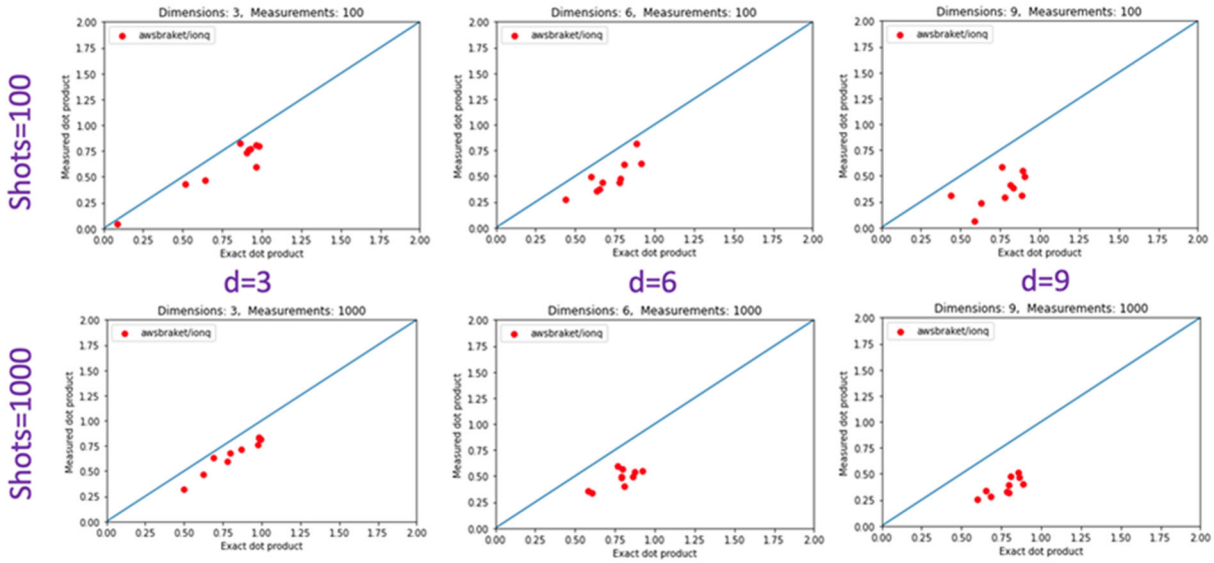


Figure 16. Measurement of the quantum dot product of random 2-dimensional vector pairs on the IonQ quantum computer for different dimensions and different numbers of shots.

From this we conclude that a largest problem size of d=9 with 1000 shots is likely to succeed. This is based on the intuition that the quantum measurement (shown in the bottom right) is accurate enough to discriminate between vectors that are close versus far apart. Whether this is true in practice will be determined by runs of the flight data using the Q-Means algorithm on quantum hardware. One always has the option of using a problem of lower dimensionality (which have better agreement with the exact measurement) to run on hardware, but we are interested in trying to maximize the largest problem size through the quantum hardware.

4.3.3. Best practices for using IonQ hardware

To implement the Q-Means algorithm on specific hardware, we looked into the characteristics of the IonQ quantum computer26, as it seemed the most capable of available choices. The IonQ trapped ion computer has the benefit by design that all qubits can interact with each other via quantum gates. This all-to-all connectivity avoids the need for additional gates to embed the logical quantum gate operations across disconnected qubits. A summary of the key features to consider are:

- Make use of all-to-all connectivity
- Prefer RZ gates (using IonQ compiler)
- Remove overly precise gates (RX, RY > p*10-3

- Optimize shot counts (balancing needed precision versus cost)
- Accurately account for XX, YY, ZZ gates (using IonQ compiler)

Additionally, we implemented two additional methods.

- Use symmetry-based post-selection (using sum-of-qubits conservation). The fact that the qdot quantum dot product uses unary circuits, which can only have valid results when only one of the qubits is found in the final quantum register measurement, any measurement that violates this restriction can be deleted as erroneous. This has been programmed into the qdot code and is automatically implemented.
- Gate reduction where possible. When the qdot algorithm generates a circuit for the given pair of vectors, and it is passed to AWS Braket, gate reductions that can be made. Most common is the double Hadamard gates, H, that can be removed via the identity that HH=*I*, the identity matrix. For the d=9 measurements, this reduced gate count from 113 To 66, which should significantly reduce the accumulated noise in running the circuit.

This last step of gate reduction was implemented in the following manner. We expand the loader circuit with the Ry translation within quasar. We then translate quasar to IBM's Qiskit, use Qiskit to optimize, and translate back to quasar, and then translate to AWS Braket. For example, for a 5-element loader circuit:

- The naive Braket translation had 22 steps (time slices, that is, H-gates on two different qubits counts as one time slice)
- The Braket roundtrip with basic optimization, removing successive H-gates, was 18 time slices
- The Braket roundtrip with a few more optimizations greatly reduced it down to 11 time slices; that's half the length of the original circuit.

An example run of this circuit compilation workflow is shown in the Appendix. If we run both circuits on a noiseless simulator—the original quasar circuit and the final transpiled circuit for IonQ on Braket—and compare them, we see very good agreement between the final computed state vectors.

4.4. Initial IonQ hardware runs with 100 shots

Now we are ready to run the full problem through the Q-Means algorithm. However, a major challenge remains in that commercially available quantum hardware is very cumbersome to use for iterative algorithms such as Q-Means or whenever you have many calls with different quantum circuits to run. The best situation is to have dedicated access to a quantum computer for hours at a time. This can only be made available through special arrangements with the hardware vendor. QC Ware has been able to do this on a few occasions, generally in a collaborative effort where the hardware vendor offers dedicated time in the hopes of a collaborative hardware/software effort will yield a significant publishable result. However, the situation is improving. Many quantum hardware cloud providers are creating frameworks and platform options where dedicated access can be granted or where a user can submit a bundle of circuits to be executed one after the other before the next user in the queue is granted control over the machine. For this project, QC Ware

wrote new code to make the best use of the Amazon Web Services's Braket Jobs framework. This took a bit more time than expected but did eventually deliver the ability to submit a large, iterative Q-Means job and get a result back in a reasonable time of less than a day.

After testing the workflow on AWS Braket's SV1 simulator, we submitted the N=10 problem instance (2 each of the 5 flight paths using d=9 dimensional data) to the IonQ quantum computer. The initial results are shown in Figure 17,


```
{'N': 10,
 'classical_labels': [0, 0, 1, 1, 2, 2, 3, 3, 4, 4],
 'expected_charge': 325.0,
 'expected_shots': 25000,
 'expected_tasks': 250,
 'iterations_before_convergence': 5,
 'k': 5,
 'max_iter': 5,
 'quantum_labels': [4, 3, 1, 1, 2, 2, 0, 1, 1, 1]}
```


Figure 17. The figure shows the text return from the QC Ware Q-Means script executed on AWS Braket Jobs via QC Ware's orchestration software.

N indicates number of flight path time histories of (x,y,z) data that were used in the problem to classify. The 'classical_labels' show the classification returned by the classical k-means algorithm. The 'expected_charge' shows the estimated charge in US dollars from AWS Braket for a run of this many iterative calls or tasks and using 100 shots (measurements) of each quantum circuit submitted. Given that we created 5 types of flight paths, we set the number of cluster centers to 'k'=5. An important point on this, is that increasing noise in the quantum circuits, which would be expected for our 9-qubit feature size, will generally prevent a smooth descent to a convergent result as it introduced some stochastic noise, the end result being that additional runs, which are costly, do not yield a better results. And finally, we see the classification scheme computed by Q-Means is not completely accurate, in that it returns the second and third pairs classified correctly (1, 1 and 2, 2), but the other flight paths are confused.

4.5. Initial IonQ hardware runs with 1000 shots

With the fairly promising result from the 100-shot run, we then submitted the problem to a full run using 1000 shots, which we believed from the testing of the qdot measurements would be sufficient to give us a better clustering result. The results from this initial 1000-shot run are shown in Figure 18.

```
{'N': 10,
"classical_labels": [0, 0, 1, 1, 2, 2, 3, 3, 4, 4],
"quantum_labels": [2, 3, 0, 0, 0, 0, 0, 0, 0, 0],
"expected_tasks": 350,
"expected_shots": 350000,
"expected_charge": 3605.0,
"k": 5,
"max_iter": 7,
"iterations_before_convergence": 7,
"history": [6.587370739156742, 4.785041404494093, 1.7053487491196528, 2.1221051204369825,
10.137864216716123, 7.310705062117888, 7.1678021380848405, 8.300042353738299],
"inertia": 8.300042353738299}
```


Figure 18. The figure shows the text return from the QC Ware Q-Means script executed on AWS Braket Jobs via QC Ware's orchestration software.

N indicates number of flight path time histories of (x,y,z) data that were used in the problem to classify. The 'classical_labels' show the classification returned by the classical k-means algorithm. The 'expected_charge' shows the estimated charge in US dollars from AWS Braket for a run of this many iterative calls or tasks and using 1000 shots (measurements) of each quantum circuit submitted.

The quantum labels in this 1000-shot result are actually worse than in the 100-shot result, which was unexpected. Additionally, when we look at the inertia of the solution ("inertia" means the residual distances summed over all points to their assigned cluster centers), we see that we are not getting smooth convergence, as shown in **Error! Reference source not found.**. With a classical solver, one would expect fairly smooth convergence downward to the final solution. In the case of the quantum Q-Means for this problem instance, we see dramatic jumps from iteration to iteration. This is due partly to the noise in measuring distances quantumly via the qdot algorithm but might be exacerbated by the small problem size with such few data points do anchor the cluster centroids. Noticing that the run took place over a two-day period, one could also suspect variable performance of the quantum computing hardware, but this is difficult to evaluate *a posteriori*.

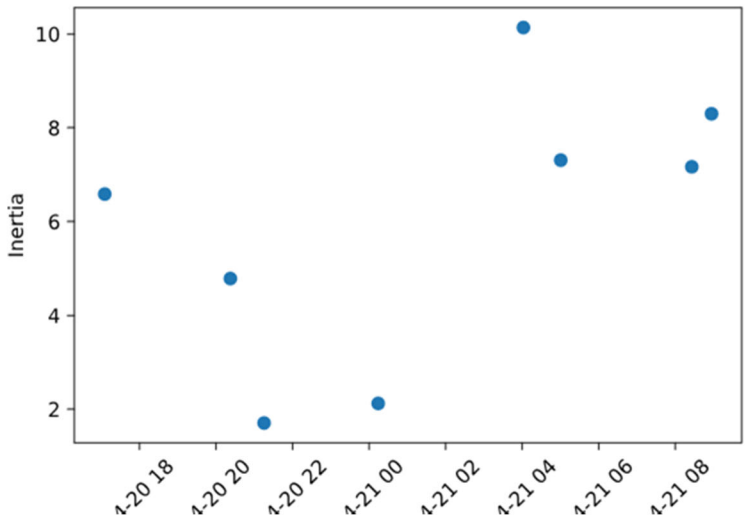


Figure 19. The figure shows the clustering inertia after each iteration of the Q-Means algorithm.

4.6. Additional Hardware Runs

We experimented with a variety of stopping conditions of the Q-Means algorithm such as maximum number of iterations and tolerance of change between successive inertia measurements. But none of these gave any consistent improvement in the results, as shown in Figure 20.

There are two challenges with these stopping conditions. For maximum iteration number, we observed that often a job would run with decreasing inertia for a few iterations, but then the code would drastically depart to a much higher level of inertia. This is likely due to the noise in the quantum measurement but might also be an artifact of the very small problem size. For tolerance in change of inertia between iterations, the noise in the measurements might be two large to allow for any sort of smooth convergence.

The 1000-shot runs appear to show lower inertia and downward convergence for early in the run but is marred by the drastic jumps away from a good solution. We used two different problem instances of the same size for these runs. Both had an optimal solution of < 0.5, so it may well be that the noise in the quantum measurement prevents convergence to an accurate measure, as the differences in inertia between successive iterations is typically larger than the minimum solution.

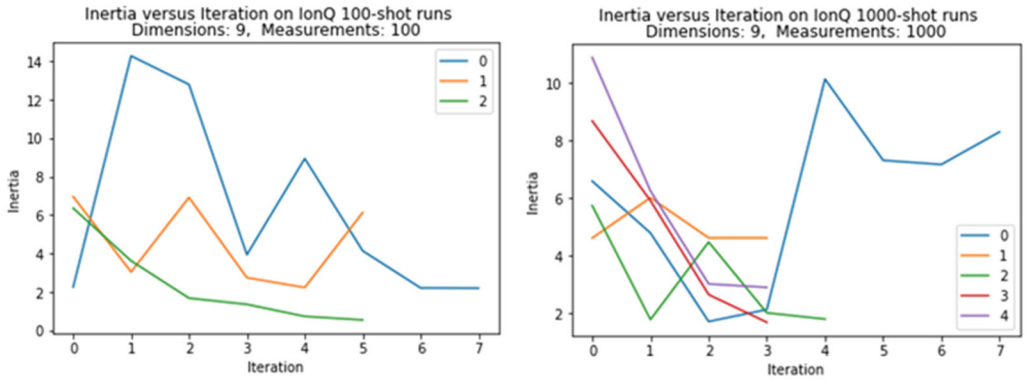


Figure 20. The figures show the progression of residual inertia for a number of hardware runs on the IonQ quantum computer. The figure on the left shows inertia histories for 100-shot runs, and the figure on the right shows the results for 1000-shot runs.

Additionally, and an important point at this stage of quantum computing technology, the 100-shot runs shown in **Error! Reference source not found.**, cost about $300.00 to run. The 1000-shot runs cost from $2000.00 to $3500.00 to run depending on how many iterations are made. To better understand this problem and the performance on hardware, significantly more problem runs would need to be made to obtain better statistics and gain additional insight. But the commercial prices for such hardware runs are quite prohibitive to do so.

4.7. Reassessing the qdot Measurements

During the last two weeks of this project, and with limited progress being made towards improving results, we repeated our initial experiment to measure the reliability of the qdot measurement on the IonQ machine. The result is in **Error! Reference source not found.**. The figures show the performance of the qdot measurement on 9-dimensional data. The plot on the left was taken on a IonQ hardware run early in the hardware experiment phase. The plot on the right was taken in the last week of the project, about one month later in time. The scales of the plots are different as the plot on the right includes a y-axis that contains negative values. The plot on the left shows moderate correlation between near and far pairs of vectors, which is hopeful for use in Q-Means, while the plot on the right gives approximately zero distance measured quantumly regardless of the actual distance, and would be unsuitable for a Q-Means run at this size of problem features.

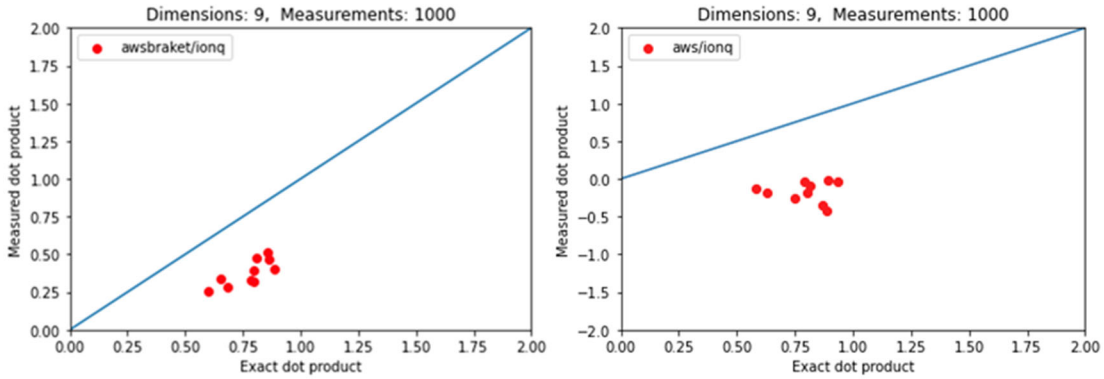


Figure 21. The figures show the performance of the qdot measurement on 9-dimensional data.

It is difficult to assess how often this might occur in the quantum hardware as there is no readily available information on how often the quantum hardware is recalibrated. Keep in mind that the Q-Means runs for even this small problem size, when submitted to the AWS Braket Jobs queue, can take anywhere from 19 hours to over 3 days, as we experienced in this project. It is quite possible that vagaries in the performance of the machine could occur during the run and corrupt the result. However, we must qualify this and say that we do not have enough information at this time to make any such assessment regarding its impact on the results of this study. But it does point out the need for future researchers to have much more insight and access to quantum hardware when conducting the performance of algorithms on real hardware.

And finally, in the course of the project we developed and implemented quantum toolkit to facilitate planning, running, and optimizing Q-Means on QC hardware to include the following features:

- Data loader
- qdot assessment tool
- Methodology for optimizing meta-parameters
  - Number of shots
  - Stopping conditions
  - Inertia/convergence tracking and reporting
- Tools to manage and streamline workflow on cloud-provided quantum hardware that are shared resources

5. CONCLUSIONS

The overall result is that our flight path problem, using even a small problem size, is still quite a challenge for contemporary quantum computing hardware. The algorithm works but has a relatively high error rate in misclassifying data. Importantly we note that one can run small diagnostic problems on just the quantum dot-product algorithm (qdot), to get a good idea of the hardware can run circuits with enough depth to get good estimations on the quantum computing hardware of choice.

6. RECOMMENDATIONS

In this performance assessment, we also came up with recommendations for future work to improve the result and to get a better understanding of the behavior of the algorithm. We summarize these in the following:

- Larger problem size (N=20 vice 10). Running the larger problem size would give better statistics on how successful the Q-Means algorithm running on hardware was. In the runs documented above, 4 or 5 out of 10 classifications were correct. A larger sample size would of course give better statistics on this.
- More shots each qdot measurement. Better statistics on how accurately a quantum processor would allow a straight-line estimation of the measured versus theoretical value of the qdot measurement. If it is repeatable, this would allow a simple modification of the qdot code to include this slope adjusting calibration as a bias reduction, thus improving the overall accuracy of the dot product and hence the distance estimation.

- Q-Means might be feasible on today's better quantum computing hardware, but for very small problems—requires significant problem sizing and parameter tuning for best performance. But it would be key to have dedicated access to the device and more insight to its calibration and performance to ensure reliable operation through the course of the experiments.
- Parallel work that should improve future performance of Q-Means and other QML algorithms
    - Quantum amplitude estimation--reduces number of shots
    - Subspace mapping—improves the speedup of quantum over classical (but requires more sophisticated circuits)

7. REFERENCES

S. Debnath, N. M. Linke, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe. Demonstration of a small programmable quantum computer with atomic qubits. *Nature*, 536(7614):63–66, 2016.

2 Thomas Monz, Daniel Nigg, Esteban A. Martinez, Matthias F. Brandl, Philipp Schindler, Richard Rines, Shannon X. Wang, Isaac L. Chuang, and Rainer Blatt. Realization of a scalable shor algorithm. *Science*, 351(6277):1068–1070, 2016.

3 R. Barends, J. Kelly, A. Megrant, A. Veitia, D. Sank, E. Jeffrey, T. C. White, J. Mutus, A. G. Fowler, B. Campbell, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, C. Neill, P. O'Malley, P. Roushan, A. Vainsencher, J. Wenner, A. N. Korotkov, A. N. Cleland, and John M. Martinis. Superconducting quantum circuits at the surface code threshold for fault tolerance. *Nature*, 508(7497):500–503, 2014.

4 A. D. Corcoles, Easwar Magesan, Srikanth J. Srinivasan, Andrew W. Cross, M. Steffen, Jay M. Gam- betta, and Jerry M. Chow. Demonstration of a quantum error detection code using a square lattice of four superconducting qubits. *Nature Communications*, 6(1):6979, 2015.

5 Norbert M. Linke, Dmitri Maslov, Martin Roetteler, Shantanu Debnath, Caroline Figgatt, Kevin A. Landsman, Kenneth Wright, and Christopher Monroe. Experimental comparison of two quantum com- puting architectures. *Proceedings of the National Academy of Sciences*, 114(13):3305–3310, 2017.

6 Prakash Murali, Norbert Matthias Linke, Margaret Martonosi, Ali Javadi Abhari, Nhung Hong Nguyen, and Cinthia Huerta Alderete. Full-stack, real-system quantum computer studies: Architectural comparisons and design insights. In *Proceedings of the 46th International Symposium on Computer Architecture*, ISCA '19, page 527, New York, NY, USA, 2019. Association for Computing Machinery.

7 Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew P. Harrigan, Michael J. Hartmann, Alan Ho, Markus Hoffmann, Trent Huang, Travis S. Humble, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul V. Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandr`a, Jarrod R. McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John C. Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin J. Sung, Matthew D. Trevithick, Amit Vainsencher, Benjamin Villalonga, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John M. Martinis. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

8 Yohichi Suzuki, Shumpei Uno, Rudy Raymond, Tomoki Tanaka, Tamiya Onodera, and Naoki Yamamoto. Amplitude estimation without phase estimation. *Quantum Information Processing*, 2020.

9 Tomoki Tanaka, Yohichi Suzuki, Shumpei Uno, Rudy Raymond, Tamiya Onodera, and Naoki Yamamoto. Amplitude estimation via maximum likelihood on noisy quantum computer. *arXiv:2006.16223*, 2020.

10 Dmitry Grinko, Julien Gacon, Christa Zoufal, and Stefan Woerner. Iterative quantum amplitude estimation. *arXiv:1912.05559*, 2019.

11 Scott Aaronson and Patrick Rall. Quantum approximate counting, simplified. *SIAM Symposium on Simplicity in Algorithms*, 2020.

12 Adam Bouland, Wim van Dam, Hamed Joorati, Iordanis Kerenidis, and Anupam Prakash. Prospects and challenges of quantum finance. *arXiv:2011.06492*, 2020.

13 Omnia Ossama, Hoda M.O. Mokhtar, and Mohamed E. El-Sharkawi. An extended k-means technique for clustering moving objects. Egyptian Informatics Journal, 12(1):45–51, Mar 2011.

14 Hamid Mokhtarzadeh and Todd Colten. Small uav position and attitude, raw sensor, and aerial imagery data collected over farm field with surveyed markers. http://dx.doi.org/10.13020/D6BC7Z, 2015.

15 Sonika Johri, Shantanu Debnath, Avinash Mocherla, Alexandros Singh, Anupam Prakash, Jungsang Kim, and Iordanis Kerenidis. Nearest centroid classification on a trapped ion quantum computer, *arXiv:2012.04145v1*, 8 Dec 2020, 2020.

16 Iordanis Kerenidis. A method for loading classical data into quantum states for applications in machine learning and optimization. U.S. Patent Application No. 16/986,553 and 16/987,235, 2020.
17 K. Wright et al. N. Grzesiak, R. Blumel. Efficient arbitrary simultaneously entangling gates on a trapped-ion quantum computer. *Nat Commun*, 11.
18 Edward Farhi and Hartmut Neven. Classification with quantum neural networks on near term processors. *arXiv:1802.06002*, 2018.
19 Edward Grant, Marcello Benedetti, Shuxiang Cao, Andrew Hallam, Joshua Lockhart, Vid Stojevic, Andrew G. Green, and Simone Severini. Hierarchical quantum classifiers. *npj Quantum Information 4*, *arXiv:1804.03680*, 2018.
20 Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Architectures for a quantum random access memory. *Phys. Rev. A* 78, 052310, 2008.
21 Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. *Phys. Rev. Lett.* 100, 160501, 2008.
22 Srinivasan Arunachalam, Vlad Gheorghiu, Tomas Jochym-O'Connor, Michele Mosca, and Priyaa Varshi- nee Srinivasan. On the robustness of bucket brigade quantum ram. *New Journal of Physics*, Vol. 17, No. 12, Pp. 123010, 2020.
23 Sergi Ramos-Calderer, Adrian Perez-Salinas, Diego Garcıa-Martin, Carlos Bravo-Prieto, Jorge Cortada, Jordi Planaguma, and Jose I. Latorre. Quantum unary approach to option pricing. *arXiv:1912.01618*, 2019.
24 Diogo Cruz, Romain Fournier, Fabien Gremion, Alix Jeannerot, Kenichi Komagata, Tara Tosic, Jarla Thiesbrummel, Chun Lam Chan, Nicolas Macris, Marc-Andre Dupertuis, and Clement Javerzac-Galy. Efficient quantum algorithms for ghz and w states, and implementation on the ibm quantum computer. *Adv. Quantum Technol.* 1900015, 2019.
25 Iordanis Kerenidis and Anupam Prakash. Quantum Recommendation Systems. 67:49:1–49:21, 2017
26 Best practices for using IonQ hardare. Found online at https://ionq.com/best-practices

APPENDIX: SOFTWARE

The software has been successfully implemented and made available as python jupyter notebooks. The repository will be uploaded to the AFRL/RI secure service and will also be made accessible via QC Ware's forge platform.

The development of the quantum software is modeled after one of the most popular classical ML libraries, called scikit-learn (https://scikit-learn.org/), where a classical version of the Nearest Centroid algorithm is available. The Use-Case model and the Q-Means algorithm and necessary subroutines are included in the repository delivered to AFRL.

The flight path use case problem is simply defined in the program inputs section of the notebook as shown in **Error! Reference source not found.**

```
################################
###### Define problem here ######
################################

n_paths  = 100   # number of paths for each of the five flight path types
n_points = 100 # number of data points per path time increment (>= 4)
variance = 1.0 # spread in speed and altitude for each class,
               # set between (0.0, 0.99) gives no variance or full variance of 2-sigma

################################

n_clusters = 5  # this value is fixed to equal the number of classes in qmeans_generate.py

################################
```

Figure 22. Entering the flight path use case inputs into the jupyter-notebook script.

And the results of the classification of the flight paths by the classical k-means and the quantum q-means are shown at the end of the notebook script execution, as shown in Figure 23.

The q-means algorithm first classically fits the model. For the prediction it calls a function distance- estimation for each centroid and each data point. The distance-estimation function runs the procedure we described above, using the function loader for each input and returns an estimate of the Euclidean distance between each centroid and each data point. The label of each data point is assigned as the label of the nearest centroid.

Note that one could imagine more quantum ways to perform the classification, where, for example, instead of estimating the distance for each centroid separately, this operation could happen in superposition. This would make the quantum algorithm faster but also increase the number of qubits needed. We remark also that the distance or inner product estimation procedure can find many more applications such as matrix- vector multiplications during clustering or training neural networks.

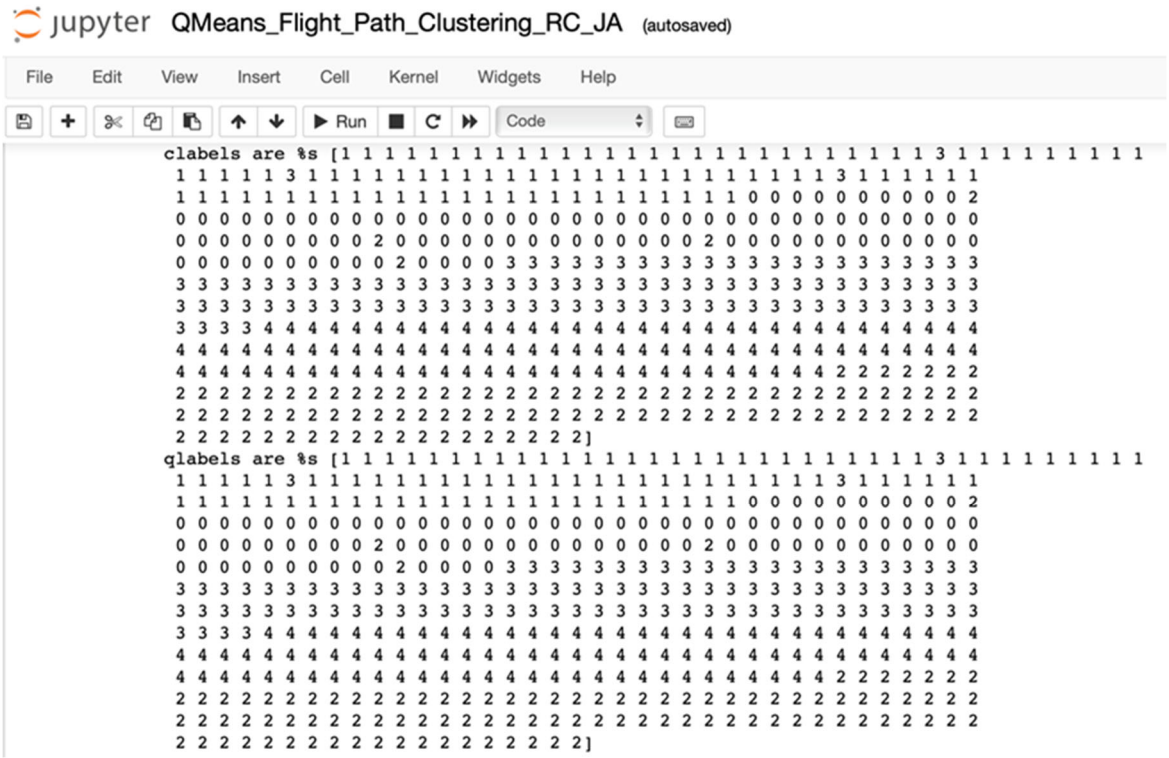


Figure 23. Displaying the classifications of the flight paths by the classical k-means and the quantum q-means algorithm after the completion of the jupyter notebook script.

An example run of this circuit compilation workflow is show in the following sequence of circuits after each phase of the process:

quasar version

```
T  : |0|1|2|3|

q0 : -X-B-B-B-
        | | |
q1 : ---|-|-S-
        | |
q2 : ---|-S-B-
        |   |
q3 : ---|---S-
        |
q4 : ---S-----

T  : |0|1|2|3|
```

expanded quasar

```
T  : |0|1|2|3 |4 |5|6|7|8|9|10|11|12|13|14|15|16|17|18|19|20|21|22|23|

q0 : -X-H-@-Ry------@-H-H-@-Ry-------@--H--H--@--Ry-------@--H--------
          |         |     |          |        |           |
q1 : -H-H-|---------|-----|----------|--------X--H--Ry-H--X--H--H-----
          |         |     |          |
q2 : -H-H-|---------|-----X-H--Ry-H--X--H--H--H--@--Ry-------@--H-----
          |         |                          |           |
q3 : -H-H-|---------|--------------------------X--H--Ry-H--X--H--H--
          |         |
q4 : -H-H-X-H--Ry-H-X-H-H-------------------------------------------

T  : |0|1|2|3 |4 |5|6|7|8|9|10|11|12|13|14|15|16|17|18|19|20|21|22|23|
```

Simplified braket

```
T  : |0|1|2|   3    |    4    |5|6|7|   8    |    9    |10|11|12|   13   |   14    |    15    |16|17|18|

q0 : -X-H-C-Ry(0.74)-------------C-C-Ry(1.15)--------------C--C--Ry(1.11)----------------------C--H-----
          |                      | |                       |  |                               |
q1 : -----|----------------------|-|-----------------------|--X--H--------Ry(-1.11)-H----------X--------
          |                      | |                       |
q2 : -----|----------------------|-X-H--------Ry(-1.15)-H--X--H--C---------Ry(0.927)---------------C--H--
          |                      |                         |                                 |
q3 : -----|----------------------|---------------------------------X--------H---------Ry(-0.927)-H--X-----
          |                      |
q4 : -----X-H--------Ry(-0.74)-H-X-------------------------------------------------------------------------

T  : |0|1|2|   3    |    4    |5|6|7|   8    |    9    |10|11|12|   13   |   14    |    15    |16|17|18|
```

Simplified braket with transpilation

```
T  : |    0    |1|   2    |3|4|   5    |6|7|   8    |    9    |10|11|

q0 : -Ry(-1.57)-C-Ry(0.74)-C-C-Ry(1.15)-C-C-Ry(1.11)-C---------H-----
                |          | |          | |         |
q1 : -----------|----------|-|----------|-X-Ry(1.11)-X---------------
                |          | |          |
q2 : -----------|----------|-X-Ry(1.15)-X-H-C--------Ry(0.927)-C--H--
                |          |               |                 |
q3 : -----------|----------|---------------X--------Ry(0.927)-X-----
                |          |
q4 : -----------X-Ry(0.74)-X---------------------------------------

T  : |    0    |1|   2    |3|4|   5    |6|7|   8    |    9    |10|11|
```

If we run both circuits on a noiseless simulator—the original quasar circuit and the final transpiled circuit for IonQ on Braket—and compare them, we see very good agreement between the final computed statevectors.

Computed quasar statevector

```
[0.         +0.j 0.67419986+0.j 0.53935989+0.j 0.         +0.j
 0.40451992+0.j 0.         +0.j 0.         +0.j 0.         +0.j
 0.26967994+0.j 0.         +0.j 0.         +0.j 0.         +0.j
 0.         +0.j 0.         +0.j 0.         +0.j 0.         +0.j
 0.13483997+0.j 0.         +0.j 0.         +0.j 0.         +0.j
 0.         +0.j 0.         +0.j 0.         +0.j 0.         +0.j
 0.         +0.j 0.         +0.j 0.         +0.j 0.         +0.j
 0.         +0.j 0.         +0.j 0.         +0.j 0.         +0.j]
```

Computed transpiled braket statevector

```
[0.         +0.j 0.67419986+0.j 0.53935989+0.j 0.         +0.j
 0.40451992+0.j 0.         +0.j 0.         +0.j 0.         +0.j
 0.26967994+0.j 0.         +0.j 0.         +0.j 0.         +0.j
 0.         +0.j 0.         +0.j 0.         +0.j 0.         +0.j
 0.13483997+0.j 0.         +0.j 0.         +0.j 0.         +0.j
 0.         +0.j 0.         +0.j 0.         +0.j 0.         +0.j
 0.         +0.j 0.         +0.j 0.         +0.j 0.         +0.j
 0.         +0.j 0.         +0.j 0.         +0.j 0.         +0.j]
```

LIST OF ACRONYMS

API	Application Programming Interface
CPU	Central Processing Unit
NASA	National Aeronautics and Space Administration
NISQ	Noisy Intermediate-Scale Quantum Computing
QML	Quantum Machine Learning
QPU	Quantum Processing Unit
QRAM	Quantum Random Access Memory
RBS	Reconfigurable Beam Splitter quantum gate
UAV	Unmanned Aerial Vehicle
USAF	United States Air Force