QUANTUM MACHINE LEARNING: STATE-OF-THE-ART REPORT AND HOW-TO GUIDE (PROVISIONAL TITLE)

September 2, 2021

Joseph Tedds - joseph.tedds@astrazeneca.com
Domingo Salazar - domingo.salazar@astrazeneca.com

September 2, 2021

1 Executive Summary

The 21st century has brought us into a quantum revolution, much like the digital revolution and the recent machine learning influx, we hope to be able to utilise these to aid our work.

- **Quantum Computing -** algorithms for quantum computing are theoretically able to provide exponential speed-up for certain problems. We see current uses for example in optimisation, machine learning and drug discovery, but may have further undiscovered uses.
- **Quantum Sensing and Metrology -** In developing quantum technology, we gain side benefits from the technology in our ability to accurately measure small phenomena. We can use these for more accurate atomic clocks, the detection of magnetic field vectors, optical phase measurement and quantum state discrimination.
- **Quantum Internet -** One of the end goals for quantum computing is to complement the classical internet with the quantum internet. It should be able to provide secure communication, secure identification, position verification, secured dedicated computing with security based on the laws of physics rather than 'hard' mathematical problems.

What recent advances makes us think that quantum computing could be useful to AstraZeneca?

- **Shor's Algorithm** [1] appearing in 1994 was arguably the beginning of quantum hype. This provides an exponential speed-up in factoring large numbers and computing discrete logarithms. Both of these problems are important in cryptography as they provide security for a number of protocols still in use today, e.g. RSA.
- Preskill [2] defines **quantum supremacy** to be '*when we will be able to perform tasks with controlled quantum systems going beyond what can be achieved with ordinary digital computers.*' In 2019, Google claimed to have achieved this [3], which was shortly refuted by IBM [4] in a blog post maintaining that the calculation could be performed in $2\frac{1}{2}$ days on a classical supercomputer. Whatever your opinion on the authenticity of this claim, this was definitely a landmark moment.
- With many high profile companies now competing in the hardware space: Google, IBM, Rigetti, large breakthroughs are being made in the numbers of qubits available and error rates. With a wide variety of approaches, companies expect to break the **threshold of 1000 qubits** some time in the next four years.
- An increasing number of companies are also providing access to quantum computers **via the cloud**. AstraZeneca already has access Microsoft Azure Quantum, but Amazon, IBM, Google, D-Wave and Oxford Quantum Circuits all have offerings.
- Very recent research by IBM [5] shows a **theoretical quantum advantage** for machine learning problems given classical input. This is important since most existing quantum advantage relied on the input of quantum data, or classical data already existing in a 'nice' quantum form.
- No universal software exists for quantum computers, just many Python libraries and one new Quantum Language, Q#.

What challenges does quantum computing face in providing practical quantum computers and usage? A by no means exhaustive list, can be broken down into three areas, with the challenges needing to be addressed by different areas of the industry.

- Hardware:
  - Noise causing errors in qubits.
  - Scaling quantum computers from tens of qubits to thousands of qubits.
  - Limited numbers of operations performed per circuit.
  - Energy costs due to cooling, close to 0K for some computers.

- Software:
  - Educating programmers on available software and algorithms.
  - Quantum computing theory outpaces current hardware, requiring clarity over what algorithms can be run on what devices.
- Uptake:
  - Costs are high, though ameliorated by cloud computing offerings.
  - Understanding the use cases e.g. for drug discovery, optimisation and machine learning.
  - Access to quantum computers.

Contents

Part I
Quantum Computing: State of the Art

In this section, we discuss the current state of quantum computing, building up some basics for comparing quantum computers and how they work. We'll also discuss the most pressing issues for implementing quantum computing, as well as potential timelines for when major milestones may occur.

2 The Basics of Quantum Computing

Quantum computing is built on linear algebra, there's no way of getting around it entirely, but we'll do the best we can. If you're interested in the more rigorous treatment, we'll revisit the basics in Part II.

In classical computing, the most basic element is a bit, taking the value either 0 or 1. We can add them together, combine them into strings and use classical gates to transform bit strings. In quantum computing, we'd like to build on these ideas, but also introduce the key quantum properties of **superposition** and **entanglement**.

In quantum computing, the most basic element is a qubit, which when measured will give the value 0 or 1. It's easier to consider a classical example first. Think of spinning a coin on its edge, at this stage it doesn't have the value heads or tails, we can only talk about the probability that it lands heads or tails. Measurement, in the form of stopping the coin, collapses the superposition and allows us to read an outcome as either heads or tails. It's exactly the same with qubits. Before we measure a qubit, they can be in a **superposition** of the states 0 and 1 and measurement causes the superposition to collapse and gives an outcome of 0 or 1 (with a probability depending on the preparation of the state).

Moving onto **entanglement**, we now need to consider multiple linked coins. The simplest example is two linked coins. We spin two coins and when we measure one, the other coin becomes fixed to the same value, i.e. we always obtain either head, head or tails, tails. Entanglement is not dependent on distance, so you can create entangled qubits and separate them. In 2017, Chinese scientists successfully maintained entanglement across 1200 km [6].

Quantum entanglement is necessary for quantum advantage, as any calculations without entanglement can be efficiently simulated by a classical computer.

Typically, we model classical computing as a series of logic gates. In quantum computing, we have a similar circuit model that uses quantum gates, but with the caveat that they must be reversible.

3 What makes a good quantum computer?

There are various different types of quantum computer, depending on how the qubits are built. We will focus only on the types of computer that are currently under development. In looking at viable quantum computers, there are a number of properties we wish to have - the DiVincenzo Criteria [7].

- **A scalable quantum system with well characterised qubits**. Good candidates for qubits are quantum 2-level systems, mimicking our coin, e.g. the up and down spin states of a spin 1/2 particle or the ground and excite states of an atom. Qubits must be able to occupy a continuum of states, and able to be entangled with one another. Well characterised also refers to its physical parameters being accurately known: the internal Hamiltonian, the presence of and coupling to other states of the qubit, the interactions with other qubits and the coupling to external fields that might be used to manipulate the state of the qubit. In the event that the system has more than 2 levels, we require that the probability of transitioning to these higher levels is very small.

- **The ability to initialise the state of the qubits to a simple fiducial state**. All computations have to start somewhere, typically we choose the simple state to be $|0 \cdots 0\rangle$. Furthermore, quantum error correction will require a continuous supply of qubits in a low entropy state, thus we require the ability to produce lots of these states accurately, and on demand. Note that this can be achieved by measuring states mid-algorithm and resetting them, if necessary, to the initial state.

- **Long relevant decoherence times, much longer than the gate operation times**. Decoherence is the process of qubits losing information due to interactions with the environment. Returning to our coin example, this is similar as letting the coin stop spinning by itself. Amplitude damping is one such example. Say we have $|0\rangle_A$,the ground state of a 2-level atom, $|1\rangle_A$ the excited state, and $|0\rangle_E$ the environment - initially in a vacuum state. If our system is in the excited state $|1\rangle_A$, then under amplitude damping it decays to $|0\rangle_A$ with probability p and spontaneous emission of a photon. In particular, we can see the decoherence of the state

$$|1\rangle_A |0\rangle_E \rightarrow \sqrt{1-p}\, |1\rangle_A |0\rangle_E + \sqrt{p}\, |0\rangle_A |1\rangle_E .$$

  In general, the decoherence time will be more complex. Thus, we see in order to perform quantum calculations, we need as long a decoherence time as possible in order to obtain reliable results.

- **A "universal" set of quantum gates**. In order to implement any general circuit, we need to be able to implement a set of gates that can generate any other gate - or a good approximation to them, i.e. with arbitrary accuracy. In classical computing, the gate set {AND, OR, NOT} is universal. One such universal gate set for quantum computing is $\{H, CX, S, T\}$, the Hadamard, CNOT, Phase and $\pi/8$ gates. Note that the choice of universal gate set is usually determined by the choice of quantum computer, as some gates are easier to implement on one type of computer than another.

- **A qubit-specific measurement capability**. To be able to complete computations, we need to be able to read the results and know which qubits the measurements correspond to. These measurements should not cause a disturbance to any other qubits in the computer and be independent of all other physical parameters of the system. Further, we wish for these measurements to be accurate in the sense that when measuring a state with density matrix given by

$$\rho = p\,|0\rangle\langle 0| + (1-p)\,|1\rangle\langle 1| + \alpha\,|0\rangle\langle 1| + \alpha^*\,|1\rangle\langle 0| ,$$

  we expect to see the result 0 with probability p, and 1 with probability $1-p$.

4 Logical and Physical Qubits

When referring to qubits, we have been assuming that the systems are noise free. They are an ideal implementation of qubits that have long enough coherence time to be useful and are resistant to errors caused by low gate fidelity. Physical qubits are simply our actual implementation of the qubits for a given system. We can use quantum circuits to create logical qubits from our physical qubits via error correcting methods.

Quantum error correction is a rich part of quantum information theory, and a way to protect against gate errors and decoherence. In classical computing, for example, we can use the repetition code to protect against errors:

$$0 \rightarrow 000, \quad 1 \rightarrow 111.$$

We can then decide the output by majority voting, i.e. if we see 010 then we conclude the input was 0. Quantum information has some subtleties attached to it that mean we can't simply clone states, and so a direct implementation of the repetition code is not possible. Instead, we can do things like entangling qubits with ancillary (extra) qubits. In general, far more complex error correcting schemes will be needed.

The amount of physical qubits needed to encode a logical qubit is dependent on the errors in the quantum system and the encoding system used. Sevilla and Riedel [8] define this more rigorously, with the number of **generalised logical qubits** able to be used as a comparative metric for different computers. Note that the only experimental reliance, here, is for the error rate of the two-qubit gates.

$$N_L = N_p f_{\text{QEC}}, \quad f_{\text{QEC}} = \left[4\frac{\log(\sqrt{10}p_P/p_L)}{\log(p_{th}/p_P)} + 1 \right]^{-2} .$$

Where:

- N_L is the number of logical qubits.

- N_P is the number of physical qubits.
- p_P is the two-qubit gate error. (We don't consider the error from gates acting on one qubit, as the two-qubit error rate is far higher).
- p_{th} is an approximate threshold error at which fault-tolerance becomes possible (see below for more on this).
- p_L the two-qubit error for the logical qubits.

We are free to pick p_L to be as we like, though since $1/p_L$ is on average the number of two-qubit gates we can apply before an error takes place, we want to choose this to be small enough to accomplish most algorithms. For a particular error correcting code, the surface code, Javadi-Avhari [9] gives $p_{th} \approx 10^{-2}$. Note that this formula is only an approximation.

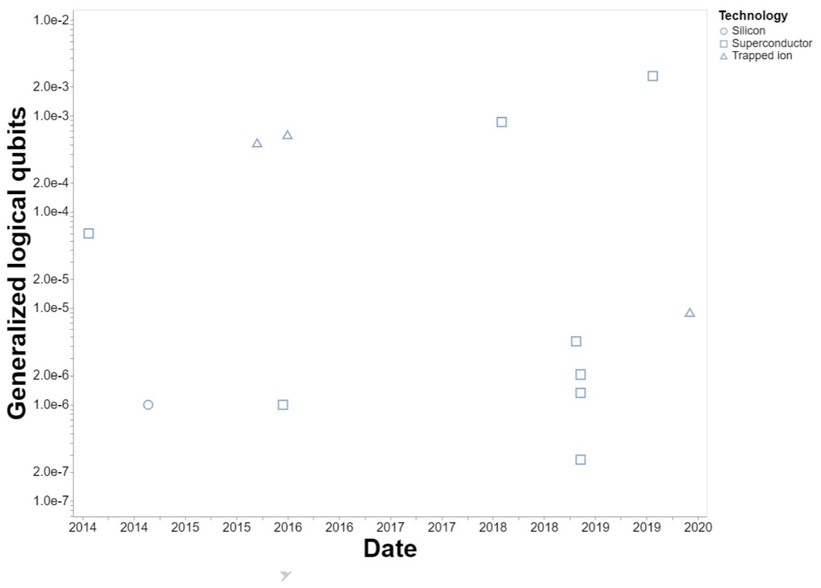


Figure 1: Calculated numbers of generalised logical qubits from 2003 to 2020 (including only those meeting the fault-tolerance threshold). In: *Towards a scalable software stack for resource estimation and optimization in general-purpose quantum computers* Javadi-Avhari p9. Figure 4 [9]

As we can see in the figure above, the Rigetti Aspen-4 has the highest number of generalised logical qubits at roughly 0.003. Logical qubits are clearly our goal, as a near-perfect realisation of our algorithms. A significant milestone then is reaching one logical qubit. Perhaps a more interesting question is how many we need to run general purpose quantum algorithms. Roettler et al. [10] put the number of logical qubits needed to factorise a 2048-bit number at roughly 4096 for Shor's algorithm.

5 Analogue, NISQ & FTQC Computers

Grumbling and Horowitz [11] broadly define 3 different types of quantum computer.

- **Analogue Quantum Computers** - quantum computers where the dynamics of the system we use for computing is similar to the model we wish to study. We are unable to manipulate the individual steps, quantum gates, directly and can only make analogue changes to the dynamics of the quantum system. For instance, a **quantum annealer** prepares an initial state for a simple Hamiltonian and the Hamiltonian is slowly altered until it becomes the problem Hamiltonian. Since the ground state will sit in the minimum energy state, we hope to find a minimal energy solution as our final state, though there is a non-zero probability that this is an excited state. These computers are useful for sampling and optimization problems, but have problems with thermal noise and running the system too quickly can introduce errors. Note that there do exist some examples of protein folding on 2-D and 3-D lattices [12], [13].

- **Noisy Intermediate Scale Quantum computers (NISQ)** - these are the digital quantum computers we currently work with. These are characterised as digital, since we can manipulate the qubits via quantum gates that allow us to complete any theoretical computation. However, our current implementations are very noisy (as the name suggests) so we can't run thousands of gates without creating large errors, and we're restricted in the number of qubits we can access.
- **Fault Tolerant Quantum Computer (FTQC)** - Digital quantum computers using logical qubits. This is the end goal of quantum computing. Since the gate-based calculations are now error free (up to an arbitrary amount) we would be able to complete algorithms of increased complexity and depth, hence utilise the full power of quantum computing.

6 Physical Realisations

There are quite a few different methods to construct a quantum computer. We will very briefly discuss only a few of the more widely used types within NISQ computers.

- **Superconducting quantum computers** are under development by companies like Google, IBM, Rigetti and D-Wave. The quantum system is formed by small current on a chip that quantum tunnels through a Josephson Junction. These circuits are typically 2-D, but Oxford Quantum Circuits has recently announced their 3-D circuits. The qubits are realised as charge distributions, the flux of the current, or a hybrid approach. Since these are essentially microchips, similar manufacturing techniques can be used to create them, so they are very scalable. However, the superconductivity needs the chip to be cooled to below 100 mK which is both energy and space intensive. Difficulty will also come when the system is scaled up to thousands of qubits, when multiple cooling systems may be required. While the decoherence time for the qubits is typically very fast, in the order of 10 μs, gate application times are low enough to implement

- **Trapped ion computers**. Honeywell and IonQ are the leaders in this field. The quantum system is formed by the movement of the ions suspended in electromagnetic fields, with entanglement created via lasers. Ion traps typically have longer decoherence times that superconducting qubits, and allow for greater connectivity (number of nearby qubits that can interact). However, the speed at which they operate can be quite slow and adding qubits becomes more difficult due to unwanted interactions between neighbouring ions. Furthermore, cooling is also required to below 2K, which although low requires far less energy than the superconducting qubits. The records for quantum volume, another metric for comparing quantum computers, has consistently been held and improved upon by Honeywell.
- **Photonic quantum computers** are light based, relying on the polarisation of light, the frequency of light or a combination of the two. These can be operated at room temperature and have long decoherence times - in the order of seconds. However, the equipment required often takes up far more space than the chips for superconducting qubits or trapped ion manipulation. Xanadu is one of the leaders in this field, having multiple cloud accessible quantum computers. PsiQuantum claim to have a method to allow miniaturisation of the chips, and are working towards building a computer with over a million qubits in the next four years.
- **Topological quantum computing** is the least popular and so far least successful method, having no known successful implementations. Microsoft were the main proponents of this method, relying on the properties of a Majorana fermion, but their observations were later shown to be inconclusive, so little progress has been made. The principles are based on the paths of specific quasi-particles known as anyons. Since their paths are unable to cross, they form braids by winding themselves around each other. The quantum state of the system of anyons depends entirely on the topology of their trajectories. This means that noise that causing small errors in the trajectories won't cause errors.

Microsoft claimed that this approach would have little to no downside, and the low error rate should allow for a low logical qubit to physical qubit ratio.

7 Current Offerings

We'll focus mainly on the cloud offering here, as they're likely to be most relevant at the current time.

- **Microsoft Azure Quantum**. Already available internally through Microsoft Azure, this give access to trapped ion providers like Honeywell and IonQ as well as optimisation through 1Qloud and Microsoft's algorithms on quantum annealers. IonQ also provides a free quantum simulator. Microsoft has developed a quantum computing language Q#, so will require an amount of training to use, but claims to have Python integration in the works.
- **IBM Quantum Experience**. IBM's quantum computers are only available via their own site and while outwardly targeted more towards researchers and educators than industrial uses, they have their own partner programme. If you want to tinker with quantum computing in your free time, I'd highly recommend taking a look. You have free access to a number of 5 qubit devices as well as simulators. IBM uses Qiskit, a specifically developed Python library,
- **Amazon Braket**. Another cloud computing option, available through AWS. This gives access to Rigetti's Aspen 9 superconducting computer, IonQ's trapped ion computer and D-Wave's quantum annealers. Amazon provide Amazon Braket SDK, a Python library, and their notebooks come integrated with PennyLane - useful for quantum machine learning.
- **Google Quantum Computing Service**. Much like IBM, this is another service requiring approval before you can use their high-end machines. Google's computers use superconducting qubits, with software integration provided by Cirq. Cirq is an open source Python library, which also offers integration with IonQ, AQT, Pasqal and Rigetti devices.
- **Xanadu**. Another cloud computing offering, but with access to multiple photonic quantum computers. Python and PennyLane integration will be used for controlling the computers, and they recommend using Strawberry Fields, a Python Library with pre-built quantum algorithms as well as simulators - so you can get started without access to Xanadu's computers.
- **Leap**. D-wave also offers access via the cloud to its quantum computers and also has its own SDK available via Python called Ocean. This provides access to their 2000 qubit quantum annealer, so while has potential for quantum chemistry, is unlikely to be as much use for quantum machine learning.
- **Oxford Quantum Circuits**. A relatively new development in the UK. Access is available through enquiry only, and no information is given about the hardware (other than they use superconducting qubits) or the software used.

Company	Type	Designation	Qubits
IBM	Superconducting	imbq_manhattan	65
Google	Superconducting	Weber	54
IonQ	Trapped Ion	-	32
Rigetti	Superconducting	Aspen-9	31
Xanadu	Photonics	X24	24
Honeywell	Trapped Ion	System Model H1	11

Table 1: Current qubit counts for quantum computers

8 Barriers to Quantum Computing

As an emerging technology, quantum computing naturally has some barriers to entry from a corporate perspective. These can be summarised into a few key areas:

- Costs - New technology is always expensive, and quantum computers are very definitely still in the development phase. Superconducting qubits and ion traps need energy to cool the computer to extreme temperatures, and hardware costs are in the millions of dollars to outright buy quantum computers.
- Skills - Quantum computers are only as valuable as the people who can use them. Since much of the hardware is controlled via Python, software engineers, developers etc. should be able to transition fairly comfortably. There are also more courses in quantum computing being offered at universities world-wide and a number of free cloud based services offering tutorials and introduction.
- Access to quantum computers - Since quantum hardware is rapidly changing, access to a variety of different computers is necessary as new algorithms become feasible. For the cost and skill reasons outlined above, however, it seems inefficient for companies to own quantum computers unless they are actively developing hardware. Cloud based quantum computing seems to be the current solution until the computers are advanced enough to mean owning one is enough to complete a wide variety of tasks. Although, it may become practical to own a quantum annealer, e.g. for protein folding, earlier than owning a NISQ computer for machine learning.
- Understanding the use cases - With many papers and hardware developments, it's difficult to keep track of what is achievable and on what machines. For example, many quantum computing algorithms like principal component analysis make use of quantum RAM - hardware that is not yet available. Typically, many companies like D-wave and Microsoft have case studies to help gauge feasibility of current options.

9 Timelines

This section needs more detail, but may not be apparent what until we've done some programming.

There are many ways we can track progress in quantum computing. For instance, the number of physical or logical qubits, quantum volume, error rate, decoherence time. For the most part, quantum volume and qubit numbers tend to be the most widely tracked metrics.

Date	Quantum Volume	Manufacturer	Type
2020, January	32	IBM	Superconducting
2020, June	64	Honeywell	Trapped Ion
2020, August	64	IBM	Superconducting
2020, November	128	Honeywell	Trapped Ion
2021, March	512	Honeywell	Trapped Ion
2021, July	1024	Honeywell	Trapped Ion

Table 2: Quantum Volume Milestones from 2020 Onwards

It should be noted that IBM have several computers with 128 quantum volume on 27 qubits available, compared to the Honeywell computers which use 10 qubits.

From the industrial side, IBM is the most forthright with their full timelines, but IonQ also provide a timeline for so called algorithmic qubit numbers, with both of these listed in the appendix. PsiQuantum are particularly ambitious, with their aim of over a million qubits on their first computer.

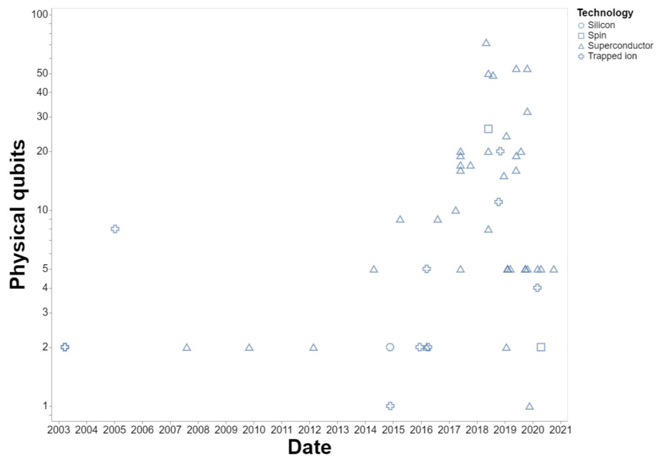


Figure 2: Physical qubit counts for 52 systems from 2003-2020. Figure from Sevilla and Riedel [8]

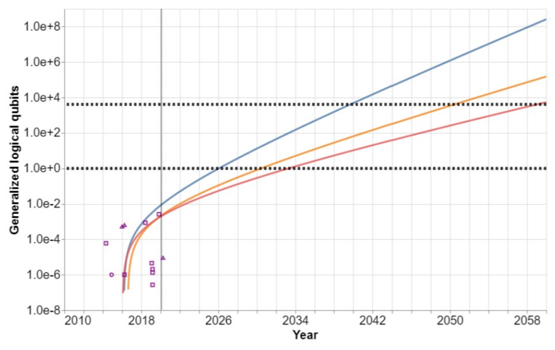


FIG. 7. Extrapolated progress of generalized logical qubits at 5%, 50%, and 95% bootstrapping quantiles (red, orange, and blue, respectively). The trend lines are based on $n = 39$ data points corresponding to superconducting quantum computers developed between 2007 to 2020 and constitute the main result of our paper. Also shown are the 12 data points from our dataset for which the number of GLQs are defined. Directly extrapolating these data points would be very unreliable because of their low number and position near the divergence at the fault-tolerance threshold.

Figure 3: Sevilla and Riedel's optimistic predictions for logical qubit progress

Part II
A Mathematical Formulation

This is where the maths is going to become a little heavier. We'll recap the basics from a firmer footing, and also touch on some places in the previous part that I'd like to have more maths in. Then we'll move on to some quantum machine learning, discuss current viable techniques and techniques that may be viable in the future.

10 Revisiting the Basics

We specified, in Part I, that the most basic element of the quantum computer was the qubit. Typically, we represent these in **bra-ket notation**, with the simplest qubits being written as

$$|0\rangle, \quad |1\rangle.$$

We call the notation for these states, kets. We extend these to linear combinations like

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle).$$

These are generalised to **superpositions**

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, |\alpha|^2 + |\beta|^2 = 1; \quad \alpha, \beta \in \mathbb{C}$$

i.e. where α, β are complex numbers that live on the surface of a sphere with radius 1. We refer to this sphere as the Bloch sphere, and it represents all possible states for a qubit.

(Bloch sphere image goes here)

Classical bits are easily measured, since a 0 is a 0 and a 1 is a 1, but our qubits are linear combinations of these 0's and 1's, so how do we measure qubits like the ones above? Measurements of a state (of a qubit) give a single outcome, with probability determined by the state, and the choice of measurement we make. In general, we will measure in the **computational basis**, where the states are represented by 0's and 1's. In the above example, we obtain 0 with probability $|\alpha|^2$ and 1 with probability $|\beta|^2$. From this, we can see that our simple states $|0\rangle$ is measured to be 0 with certainty every single time. This is exactly as we would hope for, since it is essentially a classical state. In our other example, we can see that we obtain 0 or 1 with probability 1/2.

To generalise measurements, we need to consider bras. When given a ket, we can transform it into a bra via the Hermitian transpose, i.e.

$$\langle\psi| = (|\psi\rangle)^\dagger = (\alpha|0\rangle + \beta|1\rangle)^\dagger = \alpha^*\langle 0| + \beta^*\langle 1|.$$

Now, given a state $|\psi\rangle$, we can compute the probability of obtaining a 0 as

$$p(0) = \langle\psi| P_0 |\psi\rangle = \langle\psi|0\rangle\langle 0|\psi\rangle = |\langle 0|\psi\rangle|^2 = |\alpha|^2.$$

With the products $\langle i|j\rangle = \delta_{ij}$.

Measuring the states causes the state to collapse into the measured state, so if measure a state as a 0 and repeatedly measure the state, we will always measure a 0.

Bra-ket notation is standard for quantum computing, but we can also consider this from a vectorial representation.

$$|0\rangle = \begin{pmatrix}1\\0\end{pmatrix}, \quad |1\rangle = \begin{pmatrix}0\\1\end{pmatrix}.$$

So, you can instead write state vectors as

$$|\psi\rangle = \begin{pmatrix}\alpha\\\beta\end{pmatrix}, \quad |\alpha|^2 + |\beta|^2 = 1; \quad \alpha, \beta \in \mathbb{C}.$$

Mimicking classical bit strings, we can also combine the qubits to form higher dimensional systems e.g.

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle).$$

Measuring these product states is just an extension of measuring the single qubits and in the computational basis, we can read off probabilities by squaring the amplitude of the state(s) we're interested in.

In higher dimensions, we can also introduce **entanglement**. One such example of an entangled state is the so-called EPR pair

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle).$$

It's clear that if we measure this state in the first qubit and obtain e.g a 0, then measuring the second qubit will also obtain a 0 with certainty. More generally, an n dimensional entangled state is one that can't be written as a product of n states. It's fairly simple to prove the above state is entangled by contradiction:

$$|\Phi^+\rangle = (a|0\rangle + b|1\rangle)(c|0\rangle + d|1\rangle) = ac|00\rangle + ad|01\rangle + bc|10\rangle + bd|11\rangle.$$

Now, the form of $|\Phi^+\rangle$ gives us that $ad = bc = 0$, and that $ac = bd = \frac{1}{\sqrt{2}}$. But the first condition gives us that at least two of a, b, c, d must be 0, thus contradicting the second. So we conclude that $|\Phi^+\rangle$ is entangled.

11 Quantum Volume

Note that this section is very maths heavy, a brief summary of calculating quantum volume is that the quantum computer attempts to solve a specific problem with a high probability. Success at certain depths of quantum circuits or qubit numbers are logged, and this number is transformed into a quantum volume.

To introduce quantum volume, we first need to introduce another error source and another property of quantum computers. Connectivity is, simply, the number of other qubits a qubit is connected to and hence can interact with via 2-qubit gates. Crosstalk is errors caused by unintended local interactions of qubits, either stationary or caused by gates.

Another metric for comparing quantum computers is the idea of **quantum volume**, introduced by IBM in Cross et al. [14]. This is another measure that takes into account the number of qubits available, gate errors, crosstalk and connectivity. However, this is a metric calculated by running a specific algorithm on the quantum computer, rather than being calculated directly from known qubits and error rates. Quantum volume is measured in powers of 2.

Designation	Qubits	Quantum Volume
Dublin	27	64
Kolkata	27	128
Manhattan	65	32

Table 3: Quantum Volume for IBM Computers

From the above table, it's clear that increasing the number of qubits won't necessarily increase the quantum volume. Certainly, there comes a point when the quantum volume is more important than the number of qubits available.

Quantum volume is calculated based on how well the computer can solve the **heavy output generation** problem for random model circuits. (The gates picked for the circuit are from the Haar measure over SU(4)).

The heavy set is given by

$$H_U = \{x \in \{0,1\}^m : p_U(x) > p_{\text{med}}\}, \quad p_U = |\langle x| U |0\rangle|^2,$$

and p_{med} is the median of these probabilities. The heavy output problem, then, is to generate a set of strings such that more than 2/3 of them are heavy. In practice, for a given model circuit U, the computer implements an approximation U' such that the average fidelity

$$F_{\text{avg}}(U, U') = \frac{|\text{Tr}(U^\dagger U')|^2/2^m + 1}{2^m + 1}$$

is close to 1. h_U is then our probability that an output string from U' is in the heavy set of U. We integrate h_U over all appropriate model circuits i.e over the Haar measure on SU(4) to obtain h_d.

We define the achievable model circuit depth $d(m)$ to be the largest d such that we are confident $h_d > 2/3$. Finally, we can define the quantum volume as

$$\log_2 V_Q = \text{argmax}_m \min(m, d(m))$$

i.e. the largest square shaped circuit we can implement.

It's important that we are using random circuits rather than bench-marking using very specific circuits for certain quantum algorithms, since this imitates the circuits needed to load classical data on a quantum computer.

12 Quantum Embeddings

Quantum embedding is the process of converting classical data points into quantum states, i.e. a feature map. There are several methods we can use to achieve this, each one giving rise to a corresponding kernel. Note that these embeddings (and more) are all available via the PennyLane Python library.

- **Basis Encoding -** If we are given binary inputs, we can encode strings directly into state vectors e.g.

$$010100 \to |010100\rangle .$$

  The kernel is given by the Kronecker delta δ_{ij}. This is useful for binary optimisation problems like job ordering.

- **Amplitude Encoding -** We can convert vectors straight to states by normalising them, e.g.

$$\mathbf{x} = (x_0, \ldots, x_{N_1})^T \in \mathbb{R}^N, \quad \mathbf{x} \to |\psi_{\mathbf{x}}\rangle = \frac{1}{\|\mathbf{x}\|_2} \sum_{i=0}^{N-1} x_i |i\rangle$$

  With a (normalised) linear kernel. We can extend this to a product state with d copies of the state to obtain the polynomial kernel, with power d.

- **Product Encoding -** Instead of normalising the vector, we can rescale the values into the intervals $[-\pi, \pi]$ and encode the features into the amplitudes of the qubits

$$\mathbf{x} \to |\psi_{\mathbf{x}}\rangle = \prod_{i=0}^{N-1} (\cos(x_i) |0\rangle + \sin(x_i) |1\rangle).$$

  With the kernel, the cosine kernel.

Following Lloyd et al. [15] we can extend this to

$$\mathbf{x} \to |\mathbf{x}\rangle = \Phi(x, \theta) |0 \ldots 0\rangle .$$

Where θ is a vector of physical parameters used as free variables for optimisation. In a technique called propose that adaptive training should occur on the embedding, in order to create maximum separation in the feature space. This is an example of a variational quantum algorithm - we'll see more of this later.

13 qRAM

Quantum Random Access Memory (qRAM), is the quantum version of RAM. One such architecture for this comes from Giovannetti et al. [16] and is dubbed bucket-brigade. The idea is to create a circuit that, if given addresses ψ_i on N memory cells, we can retrieve the stored memory in efficient ($O(\log N)$) time. For typical classical and quantum RAM stored in a d-dimensional lattice, you need $O(N^{\frac{1}{d}})$ switches. Given that

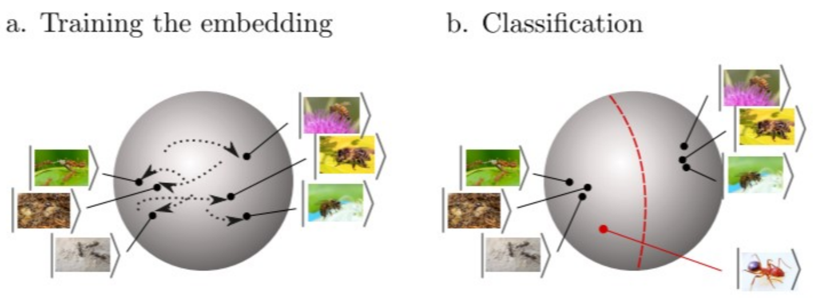


Figure 4: Quantum Metric Learning illustration from Lloyd et al. [15]

this method has time requiring exponential speedup, we see that retrieving states from this would limit the data loading parts of the algorithms.

$$\sum_i \psi_i |i\rangle \rightarrow \sum_i \psi_i |i\rangle |M_i\rangle$$

Let's first consider the classical implementation. This method requires the uses of a trit taking one of three values, wait, left and right. Any trit in the wait states will change state when it receives one bit. A value of 0 will change the state to left and 1 to right then, any incoming bits are redirected accordingly. To retrieve information, we begin the protocol will all trits in the wait state. We send the address, a bit string, through the lattice and create a path to the desired memory cell. A bus signal is then sent through the lattice, which follows the path carved out and retrieves the information from the cell. When travelling through the lattice, it resets all the trits to the wait states, thus returning the system to its initial state.

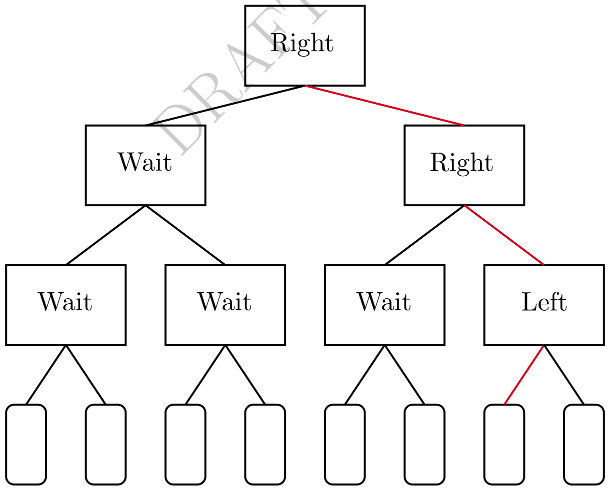


Figure 5: Bucket Brigade architecture for qRAM with 8 memory cells

With the quantum system, we replace the trits with qutrits (or qudits with $d = 3$) and similarly promote these states to $|\text{wait}\rangle, |\text{left}\rangle, |\text{right}\rangle$. At each of the nodes of the graph, if we encounter a $|\text{wait}\rangle$ we now apply a unitary transformation U

$$U|0\rangle|\text{wait}\rangle = |f\rangle|\text{left}\rangle, \quad U|1\rangle|\text{wait}\rangle = |f\rangle|\text{right}\rangle.$$

If we encounter either $|\text{left}\rangle$ or $|\text{right}\rangle$, we continue to route the address down this path. If we send a superposition of states into the lattice, by linearity we then obtain a corresponding superposition of paths through the lattice, allowing us to access the memory cells in superposition with our quantum bus. When returning, the bus applies $U^\dagger$ at each node, so that

$$U^\dagger |f\rangle |\text{left}\rangle = |0\rangle |\text{wait}\rangle , \quad U^\dagger |f\rangle |\text{right}\rangle = |1\rangle |\text{wait}\rangle$$

and the lattice is returned to its original state. Taking $N = 8$, we could for example send the code

$$|\psi\rangle = \frac{1}{\sqrt{3}}(|101\rangle + |011\rangle + |000\rangle),$$

which would return an equal superposition of the states in 0th, 3rd and 5th memory cells.

However, progress with qRAM has been slow. Consider that in order to use qRAM we still require $O(N)$ qutrits, so even storing 8 states requires a minimum of 12 qudits to retrieve a basic quantum state. (In fact precisely N qutrits, $\log N$ qubits to access the memory and an ancilla the size of the state being accessed). Thus, to effectively use qRAM for machine learning this implementation would likely need thousands of qubits. An alternative approach by Jiang et al. [17] has shown a proof-of-concept realisation of 105 qubits, stored in 210 memory cells.

14 Quantum Machine Learning

There are a number of machine learning algorithms in classical computing that have quantum counterparts. The first stage of most of these algorithms is embedding. Typically, this is achieved either by amplitude encoding or loading data from qRAM.

Far more quantum algorithms exist, than we can conceivably cover in this report, so we'll focus on a key few. You can find many of these at `https://quantumalgorithmzoo.org/`

- Quantum Principal Component Analysis [18]
- k-means Clustering [19]
- Support Vector Machines [20]
- Linear Regression [21]

Of these, on NISQ computers, the support vector machines are the only algorithms with current practical use. Quantum PCA and k-means clustering rely on qRAM or efficient state loading plus efficient Hamiltonian simulation [22].

As mentioned above, many classical machine learning algorithms have quantum counterparts, but the same is true of quantum algorithms with classical counterparts. Quantum inspired optimisation (QIO) algorithms take quantum algorithms and apply similar techniques to purely classical computations. Quantum principal component analysis has a corresponding classical algorithm also in polynomial $\log N$ time [23], although requires efficient implementation of a similar starting state to

15 Quantum PCA

The main result of LLoyd et al. [18] is the ability to construct the exponentiation of a density matrix for low rank matrices, i.e. given ρ we can construct $e^{-i\rho t}$. Quantum PCA follows from our ability to apply the quantum phase estimation algorithm using this matrix [24].

Quantum phase estimation: Given unitary operator in d dimensions and eigenstate $|v_\phi\rangle : U|v_\phi\rangle = e^{2\pi i\phi}|v_\phi\rangle$. We want to estimate the phase ϕ for $0 < \phi < 1$ to n binary bits of precision,

$$\phi \approx 0.i_1 i_2 \dots i_n = \frac{i}{2} + \frac{i_2}{4} + \dots + \frac{i_n}{2^n}, \quad i_j \in \{0, 1\}.$$

We'll need controlled U^k for integers k.

$$C - U^k |0\rangle |\psi\rangle = |0\rangle |\psi\rangle, \quad C - U^k |1\rangle |\psi\rangle = |1\rangle (U^k |\psi\rangle).$$

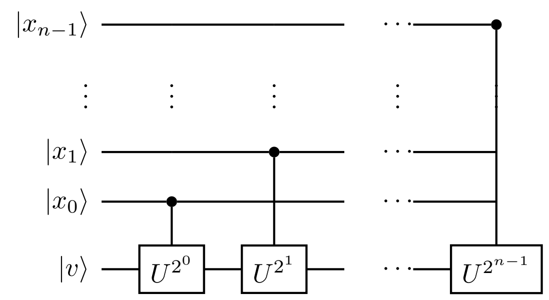


Figure 6: Circuit to implement $|x\rangle|v\rangle \to |x\rangle U^x|v\rangle$, for $x = x_{n-1}\cdots x_1x_0 = x_0+2x_1+2^2x_2+\cdots 2^{n-1}x_{n-1}, x \in \mathbb{Z}_{2^n}$

We'll actually want "generalised controlled U^k "

$$|x\rangle|\psi\rangle \to |x\rangle U^x|\psi\rangle, \quad x \in \mathbb{Z}_{2^n}.$$

Note: if input $|\psi\rangle = |v_\phi\rangle$, then we get $e^{2\pi i\phi x}|x\rangle|v_\phi\rangle$.

The general algorithm is as follows: superpose over all $x = 0, 1, 2, \ldots, 2^{n-1}$ and use $|\psi\rangle = |v_\phi\rangle$ Finally, apply QFT_{2^n} to the resulting state and measure to see $y_0, y_1, \ldots, y_{n-1}$ on lines $0, 1, \ldots, n-1$ then output the number

$$0.y_0y_1\ldots y_{n-1} = \frac{y_0}{2} + \frac{y_1}{4} + \cdots + \frac{y_{n-1}}{2^n},$$

as an estimate of ϕ. That's the phase estimation algorithm (for given U and $|v_\phi\rangle$.
Suppose ϕ actually only had n binary digits i.e. $\phi = 0.z_0z_1\ldots z_{n-1}$ for some $z_0, \ldots, z_{n-1}$ then

$$\phi = \frac{z_0z_1\cdots z_{n-1}}{2^n} = \frac{z}{2^n},$$

for z an n bit integer in $\mathbb{Z}_{2^n}$ and

$$|A\rangle = \frac{1}{\sqrt{2^n}}\sum_x e^{2\pi i\frac{xz}{2^n}}|x\rangle$$

is QFT of $|z\rangle$. So $QFT^{-1}|A\rangle = |z\rangle$ and we get ϕ exactly and with certainty.
Note: in this case, the PE algorithm up to and not including the final measurement is a unitary operation U_{PE} mapping

$$|0\rangle\ldots|0\rangle|v_\phi\rangle \to |z_0\rangle|z_1\rangle\ldots|z_{n-1}\rangle|v_\phi\rangle.$$

If ϕ has more than n bits

$$\phi = 0.z_0z_1\ldots z_{n-1}; z_nz_{n+1},$$

then we have

Theorem 15.1 (Phase Estimation)**.** If measurements in the above algorithm give $y_0, y_1, \ldots, y_{n-1}$ so output is $\theta = 0.y_0y_1\ldots y_{n-1}$ then

- $P(\theta$ is closest n binary digit approx to $\phi) \geq \frac{4}{\pi^2} \approx 0.4$
- $P(|\ \theta - \phi\ | \geq \varepsilon)$ is at most $O\left(\frac{1}{2^n\varepsilon}\right)$.

If we take a general state $|\psi\rangle$ instead, and use $U = e^{-i\rho}$ for our density matrix ρ, then we can take

$$|\psi\rangle|0\rangle \to \sum_i a_i|v_{\phi_i}\rangle|\tilde{\lambda}_i\rangle,$$

where $|v_{\phi_i}\rangle$ are the eigenvectors of ρ and $\tilde{\lambda}_i$ are the corresponding eigenvalue estimates. LLoyd et al [18] suggest using ρ as the initial state, so the algorithm gives the final state

$$\sum_i \lambda_i|v_{\phi_i}\rangle\langle v_{\phi_i}| \otimes |\tilde{\lambda}_i\rangle\langle\tilde{\lambda}_i|.$$

Sampling from this state allows us to obtain information about the eigenvalues and eigenstates of the density matrix.

16 Variational Quantum Algorithms

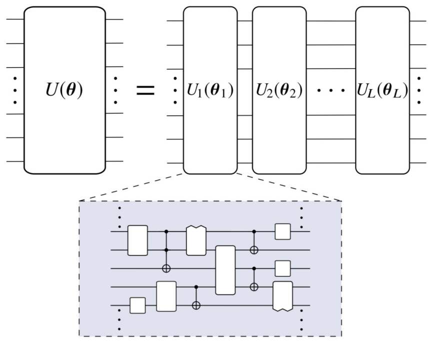
Variational quantum algorithms (VQAs) are a growing class of machine learning options. A detailed review of the field is given in [25]. These are likely to be the most useful algorithms in the near term.

VQAs tend to be most comparable to classical neural networks, and can be thought of as a class of hybrid quantum-classical algorithms.

We begin by casting the problem we wish to solve as the solution to an optimisation

$$|\Psi^*\rangle = \text{argmin}_{|\Psi\rangle} \langle\Psi| H |\Psi\rangle ,$$

such that the cost function $C(|\psi\rangle) = \langle\Psi| H |\Psi\rangle$ is a faithful representation of the problem. We construct an entangling quantum circuit of low depth - this is due to constraints on NISQ computers and log_2 of the quantum volume should give the maximum circuit depth you can use. It's also important that the circuit is entangling, else we gain no quantum advantage and this process is equivalent to a classical optimisation task.



$U(\boldsymbol{\theta})$
=
$U_1(\boldsymbol{\theta}_1)$
$U_2(\boldsymbol{\theta}_2)$
$\cdots$
$U_L(\boldsymbol{\theta}_L)$

Figure 7: Parameterised Quantum Circuit [25].

We then train an optimiser on a set of data using either parameter-shift [26] or backpropagation [27]. Note that backpropagation is only possible via simulators.

17 Existing Examples

PennyLane has a number of tutorials for quantum machine learning examples. This mostly follows `https://pennylane.ai/qml/demos/tutorial_ensemble_multi_qpu.html`, an ensemble method, classifying data from the Iris dataset. Two classifiers have been pre-trained on quantum circuits, so this is verification of parameters rather than hyper-parameter tuning. Two computations are run in parallel on quantum computers, and three qubits are measured in the computational basis. For every measurement taking value 0 we, assign it 1, and for every measurement taking value 1, we assign it -1. We then average over all such measurements, and rescale to obtain classification probabilities. For a given piece of data, we assign the class that has the highest probability from either of the two circuits, i.e. the circuit that is most certain, picks the class.

Both circuits use product encoding, with a rotation around the x-axis of the Bloch sphere.

$$R_x(\phi) = e^{-i\phi\sigma_x/2} = \begin{pmatrix} \cos(\phi/2) & -i\sin(\phi/2) \\ -i\sin(\phi/2) & \cos(\phi/2) \end{pmatrix},$$

as well as general rotation gates controlled by a set of 16 training parameters

$$\theta_{ij}^{(k)} \in [-\pi, \pi)^3; \quad i, k \in \{0, 1\}, \quad j \in \{0, 1, 2, 3\}.$$

General rotations are given by

$$R(\phi,\theta,\omega) = R_z(\omega)R_y(\theta)R_z(\phi) = \begin{pmatrix} e^{-i(\phi+\omega)/2}\cos(\theta/2) & -e^{-i(\phi-\omega)/2}\sin(\theta/2) \\ e^{-i(\phi-\omega)/2}\sin(\theta/2) & e^{i(\phi+\omega)/2}\cos(\theta/2) \end{pmatrix}$$

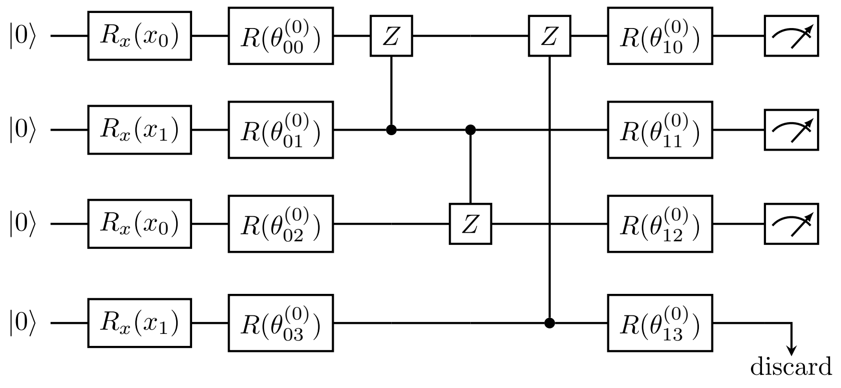


Figure 8: Device 0 Quantum Circuit

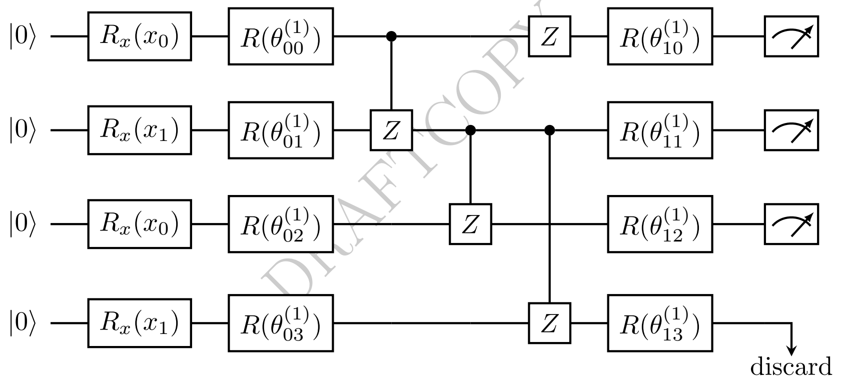


Figure 9: Device 1 Quantum Circuit

	Training	Test
Ensemble	0.8	0.64
QPU0	0.648	0.56
QPU1	0.296	0.32

Table 4: Accuracy of predictions using IBM's Manila and Santiago (5 qubit, 32 quantum volume) computers as devices 0 and 1 respectively. The training set had 125 pieces of data, and the test set 25.

Part III
Software Implementation

We now have a chance to look at some code for quantum computing, and implement a genuinely novel example.

18 The Dataset

The dataset we're considering is the protac database, available [28]. Our aim is to produce two relevant examples from this - classification and regression.

DC50 (nM)	Molecular Weight	Exact Mass	logP	logS	Heavy Atom Count	Ring Count	Hydrogen Bond Accep
96.9	991.015	990.3935	3.53	-5.02	71	7	17
11.9	787.834	787.3078	3.25	-5.04	58	8	14
276.5	1039.686	1038.376	4.25	-5	72	8	17
6.3	912.965	912.3667	3.4	-4.89	67	9	18
700	832.988	832.3479	3.56	-4.58	60	7	13

19 Classification

The classification algorithm, we are going to use, is based on a variational quantum circuit using the PennyLane library [29] in Python. The circuit functions similarly to an encoder for a classical neural network. Simply scale our input data to the interval $[-\pi, \pi]$, encode each feature as $R_X(x_i), i \in \{1, 2, \ldots, 10\}$, perform the variational circuit and put the circuit measurements into a threshold function.

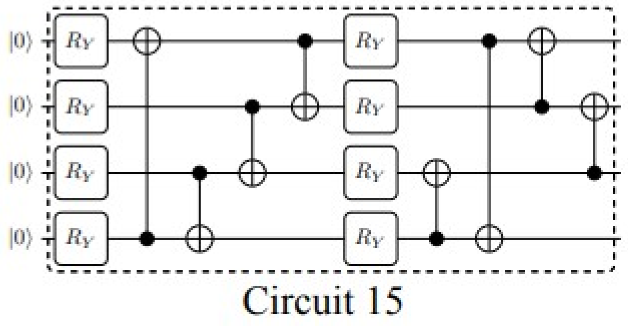


Figure 10: Circuit 15 [30]

Generating the circuit is fairly straightforward, but requires a generalisation to 10 qubits (or more) instead of 4.

```python
import pennylane as qml

def circuit15(params,x):
    n = x.size
# Requires 2n parameters, index keeps track of current parameter
    index = 0
# RY block
    for i in range(n):
        qml.RY(params[index], wires=i)
        index += 1

# CNOT ring
    qml.CNOT(wires=[n-1,0])
    index += 1
    for i in range(n-1):
        qml.CNOT(wires=[n-2-i,n-1-i])
        index +=1

# RY block
    for i in range(n):
        qml.RY(params[index], wires=i)
        index += 1

# CNOT ring
    qml.CNOT(wires=[n-1,n-2])
    index += 1
    qml.CNOT(wires=[0,n-1])
    index += 1

    for i in range(n-2):
        qml.CNOT(wires=[i+1,i])
        index +=1

    measurements = [qml.expval(qml.PauliZ(i)) for i in range(n)]
    return measurements
```

Note that the difference between this circuit and the one implemented in the model example is simply the inclusion of the angle embedding term.

```
def circuit15(params,x):
    n = x.size
    qml.templates.AngleEmbedding(x,wires=[i for i in range(n)])
    ...
```

The classifier in this case is simply a linear model with a sigmoid function. The sigmoid function returns a value between 0 and 1, so we take the model to classify 0 as the output $\sigma(y \cdot \boldsymbol{\beta} + \beta_0) \leq 0.5$.

```
def classifier(var,x,circuitNum=5):
# Pick the quantum device we're running the circuit on.
    dev = qml.device("default.qubit", wires=x.size)
    circuit = qml.QNode(circuit15,dev)
# Run the circuit
    measurements = circuit(var,x)
# Activation function
    lin_model = np.inner(measurements,var[-11:-1]) + var[-1]
    sig = 1 / (1 + np.exp((-1)*lin_model))

    return sig
```

Now for the optimisation. Since our circuit is composed of R_y and CNOT gates, we can only optimise the R_y gates. Hence we have 20 parameters (the angle of rotation about the y axis of the Bloch sphere) that we can optimise in the circuit, and 11 parameters corresponding to the linear combination of measurements.

PennyLane provides a range of optimisers - some specifically NumPy optimisers that have been altered to work with quantum circuits, and some purely quantum options. We've chosen to use the Adam optimiser. The default option is to use back-propagation to compute derivatives, which is both space and time saving

- but not possible on a quantum device. Thus, we have to use a simulator instead of a quantum computer. We could instead use the parameter-shift method, which works on all quantum computers, but also must incur an additional cost per data per shot. Note that the cost function we're using here is the squared loss function and not the log loss function because this appeared to give us better results.

```
np.random.seed(32)
var_init = (0.1*np.random.randn(parameterNum))
var= var_init

batch_size = 20
for it in range(19):
    # Update the weights by one optimizer step

    X_train_batch = XTrain[it*batch_size:(it +1)*batch_size]
    y_train_batch = yTrain[it*batch_size:(it +1)*batch_size]
    var = opt.step(lambda v: cost(v, X_train_batch, y_train_batch,circuitNum), var)
```

A very simple run of this circuit with a 4 : 1 split of training:test data and no cross validation gives us an accuracy of 80% and an f1 score of 0.095 on our test data. A simple implementation of a classical method

20 Notes

This is a selection of observations from working with PennyLane.

- When defining circuits via functions, you cannot specify the size of the circuit within the function variables. Consider the following circuit that should embed the data onto *numWires* qubits, apply a Hadamard gate to each qubit and return the expected outcome of a measurement in the Z direction.

```
    def circuit(x, numWires):
        dev = qml.device("qubit.default", wires=numWires)
        qml.templates.AngleEmbedding(x,wires=[i for i in range(numWires)])
        qml.broadcast(qml.Hadamard,[i for i in range(numWires)], "single")
        return [qml.expval(qml.PauliZ(i)) for i in range(numWires)]
```

References

[1] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM Journal on Computing* 26.5 (Oct. 1997), pp. 1484–1509. ISSN: 1095-7111. DOI: 10.1137/s0097539795293172. URL: http://dx.doi.org/10.1137/S0097539795293172.

[2] John Preskill. *Quantum computing and the entanglement frontier.* 2012. arXiv: 1203.5813 [quant-ph].

[3] Frank Arute et al. "Quantum Supremacy using a Programmable Superconducting Processor". In: *Nature* 574 (2019), pp. 505–510. URL: https://www.nature.com/articles/s41586-019-1666-5.

[4] Edwin Pednault et al. *On "Quantum Supremacy".* Aug. 2020. URL: https://www.ibm.com/blogs/research/2019/10/on-quantum-supremacy/.

[5] Yunchao Liu, Srinivasan Arunachalam, and Kristan Temme. "A rigorous and robust quantum speed-up in supervised machine learning". In: *Nature Physics* (July 2021). ISSN: 1745-2481. DOI: 10.1038/s41567-021-01287-z. URL: http://dx.doi.org/10.1038/s41567-021-01287-z.

[6] Juan Yin et al. "Satellite-based entanglement distribution over 1200 kilometers". In: *Science* 356.6343 (2017), pp. 1140–1144. ISSN: 0036-8075. DOI: 10.1126/science.aan3211. eprint: https://science.sciencemag.org/content/356/6343/1140.full.pdf. URL: https://science.sciencemag.org/content/356/6343/1140.

[7] David P. DiVincenzo. "The Physical Implementation of Quantum Computation". In: *Fortschritte der Physik* 48.9-11 (Sept. 2000), pp. 771–783. ISSN: 1521-3978. DOI: 10.1002/1521-3978(200009)48:9/11<771::aid-prop771>3.0.co;2-e. URL: http://dx.doi.org/10.1002/1521-3978(200009)48:9/11%3C771::AID-PROP771%3E3.0.CO;2-E.

[8] Jaime Sevilla and C. Jess Riedel. *Forecasting timelines of quantum computing.* 2020. arXiv: 2009.05045 [quant-ph].

[9] Ali Javadi-Avhari. *TOWARDS A SCALABLE SOFTWARE STACK FOR RESOURCE ESTIMATION AND OPTIMIZATION IN GENERAL-PURPOSE QUANTUM COMPUTERS.* July 2017. URL: http://arks.princeton.edu/ark:/88435/dsp01jq085n573.

[10] Martin Roetteler et al. *Quantum resource estimates for computing elliptic curve discrete logarithms.* 2017. arXiv: 1706.06752 [quant-ph].

[11] Engineering National Academies of Sciences and Medicine. *Quantum Computing: Progress and Prospects.* Ed. by Emily Grumbling and Mark Horowitz. Washington, DC: The National Academies Press, 2019. ISBN: 978-0-309-47969-1. DOI: 10.17226/25196. URL: https://www.nap.edu/catalog/25196/quantum-computing-progress-and-prospects.

[12] Alejandro Perdomo-Ortiz et al. *Finding low-energy conformations of lattice protein models by quantum annealing.* 2012. arXiv: 1204.5485 [quant-ph].

[13] Tomáš Babej, Christopher Ing, and Mark Fingerhuth. *Coarse-grained lattice protein folding on a quantum annealer.* 2018. arXiv: 1811.00713 [quant-ph].

[14] Andrew W. Cross et al. "Validating quantum computers using randomized model circuits". In: *Physical Review A* 100.3 (Sept. 2019). ISSN: 2469-9934. DOI: 10.1103/physreva.100.032328. URL: http://dx.doi.org/10.1103/PhysRevA.100.032328.

[15] Seth Lloyd et al. *Quantum embeddings for machine learning.* 2020. arXiv: 2001.03622 [quant-ph].

[16] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. "Quantum Random Access Memory". In: *Physical Review Letters* 100.16 (Apr. 2008). ISSN: 1079-7114. DOI: 10.1103/physrevlett.100.160501. URL: http://dx.doi.org/10.1103/PhysRevLett.100.160501.

[17] N. Jiang et al. "Experimental realization of 105-qubit random access quantum memory". In: *npj Quantum Information* 5 (Dec. 2019). DOI: 10.1038/s41534-019-0144-0.

[18] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. "Quantum principal component analysis". In: *Nature Physics* 10.9 (July 2014), pp. 631–633. ISSN: 1745-2481. DOI: 10.1038/nphys3029. URL: http://dx.doi.org/10.1038/nphys3029.

[19] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. *Quantum algorithms for supervised and unsupervised machine learning.* 2013. arXiv: `1307.0411 [quant-ph]`.

[20] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. "Quantum Support Vector Machine for Big Data Classification". In: *Physical Review Letters* 113.13 (Sept. 2014). ISSN: 1079-7114. DOI: `10.1103/physrevlett.113.130503`. URL: `http://dx.doi.org/10.1103/PhysRevLett.113.130503`.

[21] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. "Prediction by linear regression on a quantum computer". In: *Physical Review A* 94.2 (Aug. 2016). DOI: `10.1103/physreva.94.022342`. URL: `http://dx.doi.org/10.1103/PhysRevA.94.022342`.

[22] Seth Lloyd. "Universal Quantum Simulators". In: *Science* 273.5278 (1996), pp. 1073–1078. ISSN: 0036-8075. DOI: `10.1126/science.273.5278.1073`. eprint: `https://science.sciencemag.org/content/273/5278/1073.full.pdf`. URL: `https://science.sciencemag.org/content/273/5278/1073`.

[23] Ewin Tang. *Quantum-inspired classical algorithms for principal component analysis and supervised clustering.* 2019. arXiv: `1811.00414 [cs.DS]`.

[24] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition.* 10th. USA: Cambridge University Press, 2011. ISBN: 1107002176.

[25] M. Cerezo et al. *Variational Quantum Algorithms.* 2020. arXiv: `2012.09265 [quant-ph]`.

[26] Gavin E. Crooks. *Gradients of parameterized quantum gates using the parameter-shift rule and gate decomposition.* 2019. arXiv: `1905.13311 [quant-ph]`.

[27] Masaya Watabe et al. *Quantum Circuit Parameters Learning with Gradient Descent Using Backpropagation.* 2019. arXiv: `1910.14266 [quant-ph]`.

[28] Gaoqi Weng et al. "PROTAC-DB: an online database of PROTACs". In: *Nucleic Acids Research* 49.D1 (Oct. 2020), pp. D1381–D1387. ISSN: 0305-1048. DOI: `10.1093/nar/gkaa807`. eprint: `https://academic.oup.com/nar/article-pdf/49/D1/D1381/35363816/gkaa807.pdf`. URL: `https://doi.org/10.1093/nar/gkaa807`.

[29] Ville Bergholm et al. *PennyLane: Automatic differentiation of hybrid quantum-classical computations.* 2020. arXiv: `1811.04968 [quant-ph]`.

[30] Sukin Sim, Peter D. Johnson, and Alán Aspuru-Guzik. "Expressibility and Entangling Capability of Parameterized Quantum Circuits for Hybrid Quantum-Classical Algorithms". In: *Advanced Quantum Technologies* 2.12 (Oct. 2019), p. 1900070. ISSN: 2511-9044. DOI: `10.1002/qute.201900070`. URL: `http://dx.doi.org/10.1002/qute.201900070`.

[31] Karl Wehden, Ismael Faro, and Jay Gambetta. *IBM Quantum Development Roadmap.* 2021. URL: `https://www.ibm.com/blogs/research/2021/02/quantum-development-roadmap/`.

A Definitions and Acronyms

- FTQC - Fault Tolerant Quantum Computers
- NISQ - Noisy Intermediate Scale Quantum computers
- QAOA - Quantum Approximate Optimisation Algorithms
- QEC - Quantum Error Correction
- QIO - Quantum Inspired Optimisation
- qRAM - Quantum Random Access Memory
- VQA - Variational Quantum Algorithm
- VQE - Variational Quantum Eigensolver

B Industrial Timelines

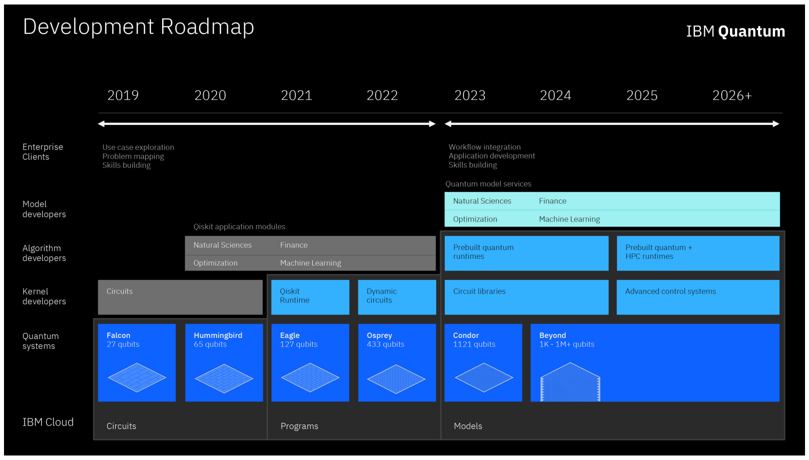


Figure 11: IBM timeline for Quantum Computing [31]

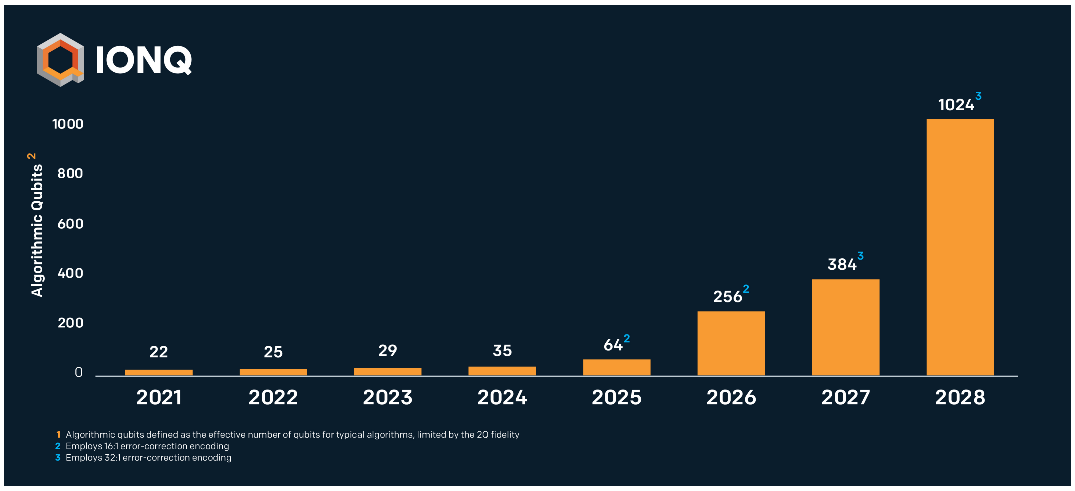


Figure 12: IonQ Qubit Roadmap