

Quantum Software



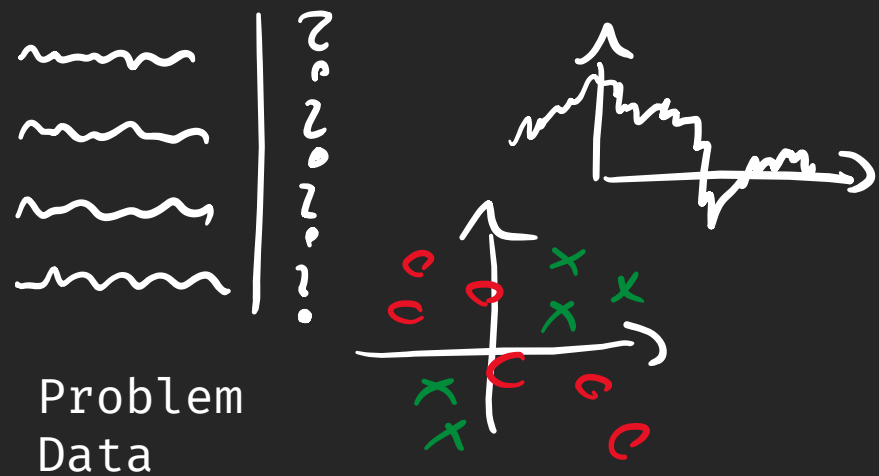
UNIVERSITY OF
OXFORD

What is Quantum Computing?

- Quantum Computers as black-box devices used inside classical algorithms
- Quantum Computers as a programmable interface to the quantum world

What is Quantum Computing?

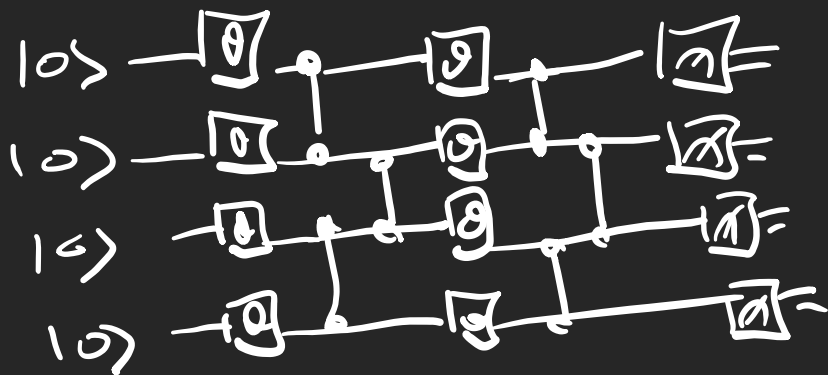
- Quantum Computers as black-box devices used inside classical algorithms
 - Breaking your Bitcoin wallet (one day)
 - Combinatorial Optimization
 - Solutions of Linear/non-linear Systems
 - Complex Physics Simulations (e.g. fluid dynamics)
 - Machine Learning
- Quantum Computers as a programmable interface to the quantum world



← Iterate (?)



↓ Pre-processing and/or encoding



↑ Post-processing and/or decoding



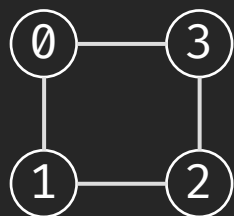
Quantum Circuit(s)

Quantum Processing Unit

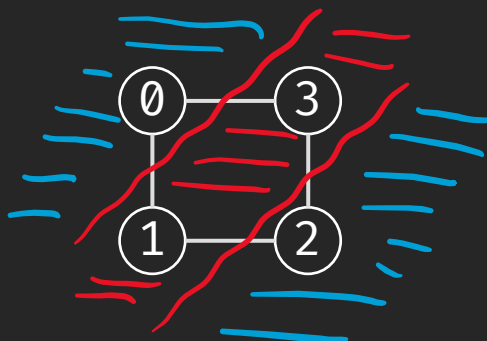
Measurement Samples

Example: max-cut with QAOA

Graph

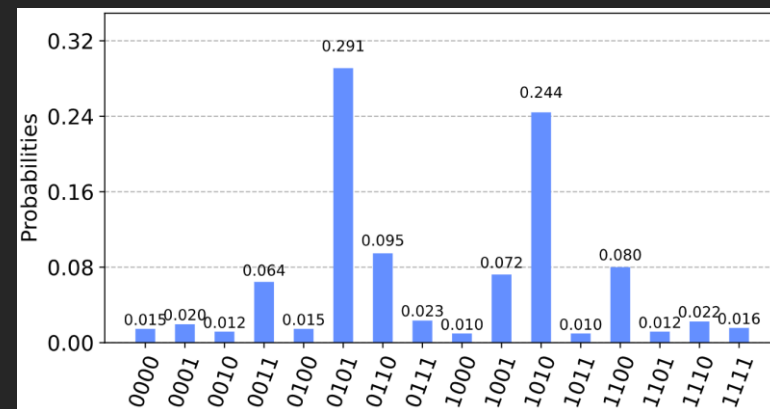


Graph Cut



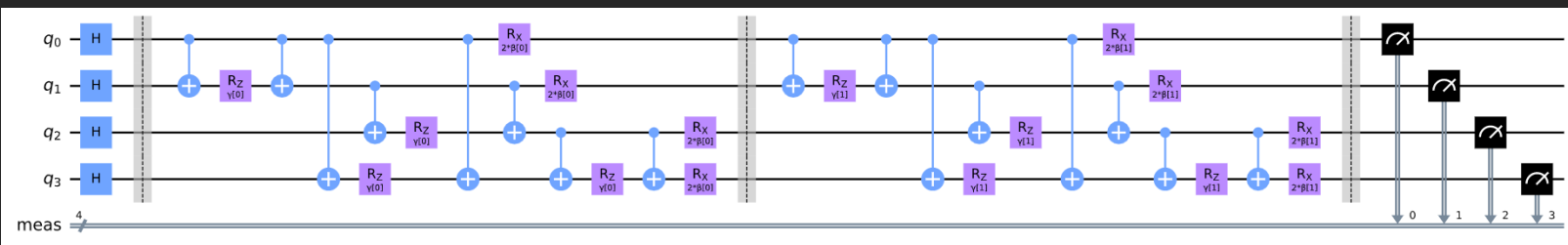
Max count solution

Samples



Encoding using 2-qubit gates

Parameter optimisation



ibmq_santiago

System status ● Online

Processor type Falcon r4

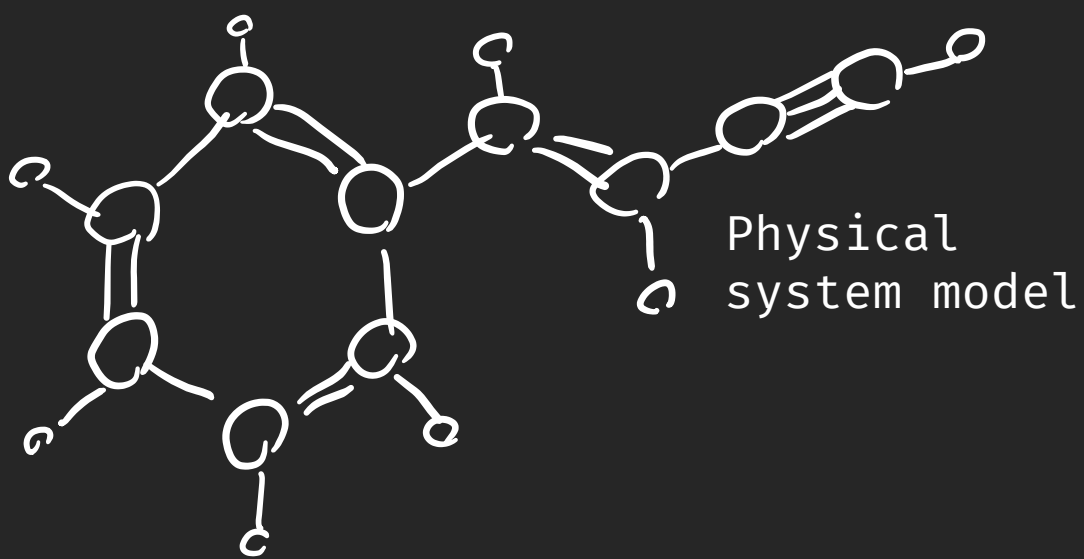
5 Qubits 32 Quantum volume

QPU

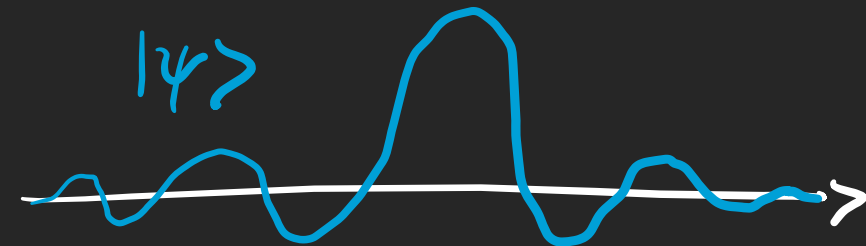
QAOA Circuit (parametric)

What is Quantum Computing?

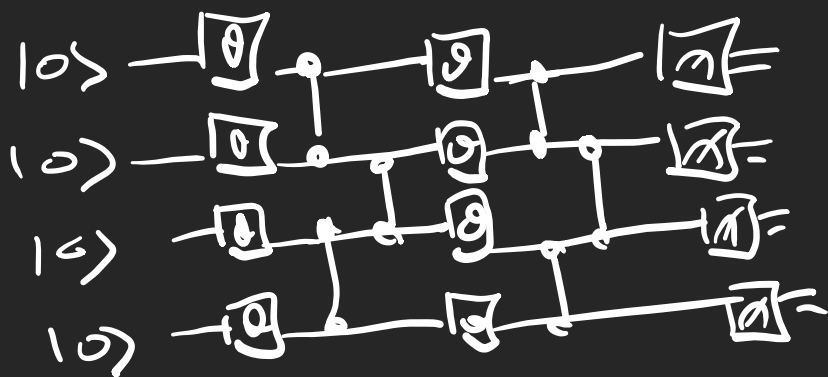
- Quantum Computers as black-box devices used inside classical algorithms
- Quantum Computers as a programmable interface to the quantum world
 - Quantum Chemistry Simulations
 - Quantum Physics Simulations
 - Quantum Information/Communication/Cryptography
 - Quantum Metrology
 - Programmable Quantum Experiments



$$\psi^{(\alpha)}(x,t) = \left(\frac{m\omega}{\pi\hbar}\right)^{1/4} \exp\left(-\frac{m\omega}{2\hbar}\left(x - \sqrt{\frac{2\hbar}{m\omega}}\Re[\alpha(t)]\right)^2 + i\sqrt{\frac{2m\omega}{\hbar}}\Im[\alpha(t)]x + i\theta(t)\right)$$



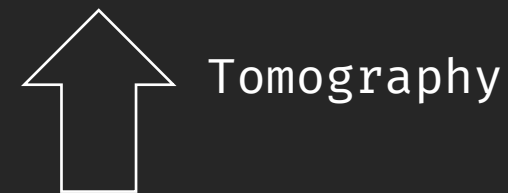
Quantum state



Quantum Circuits



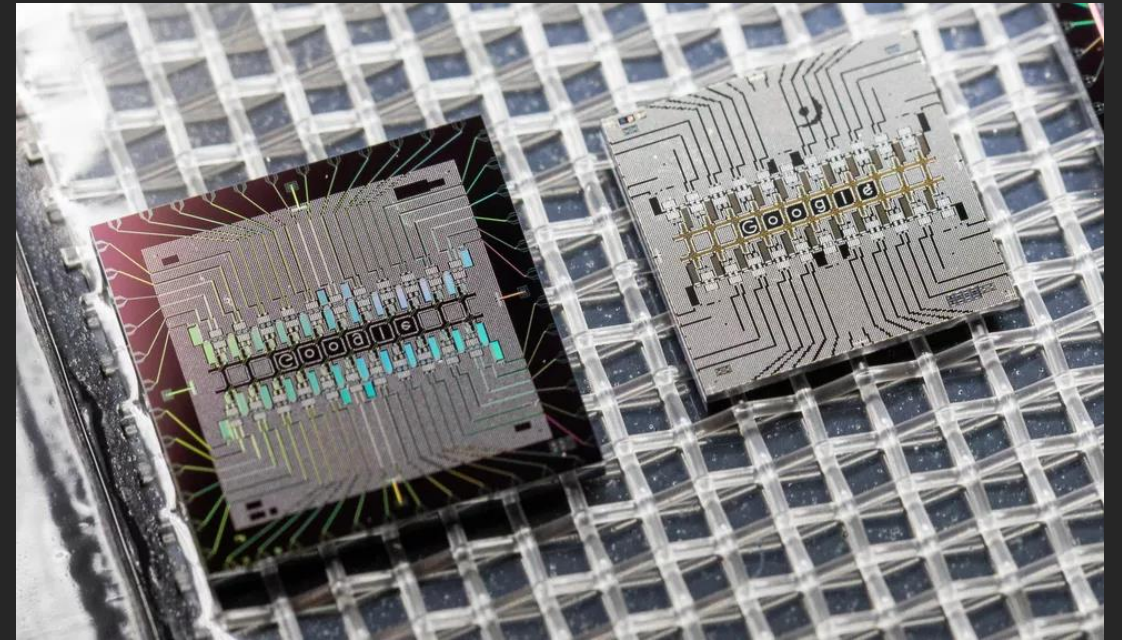
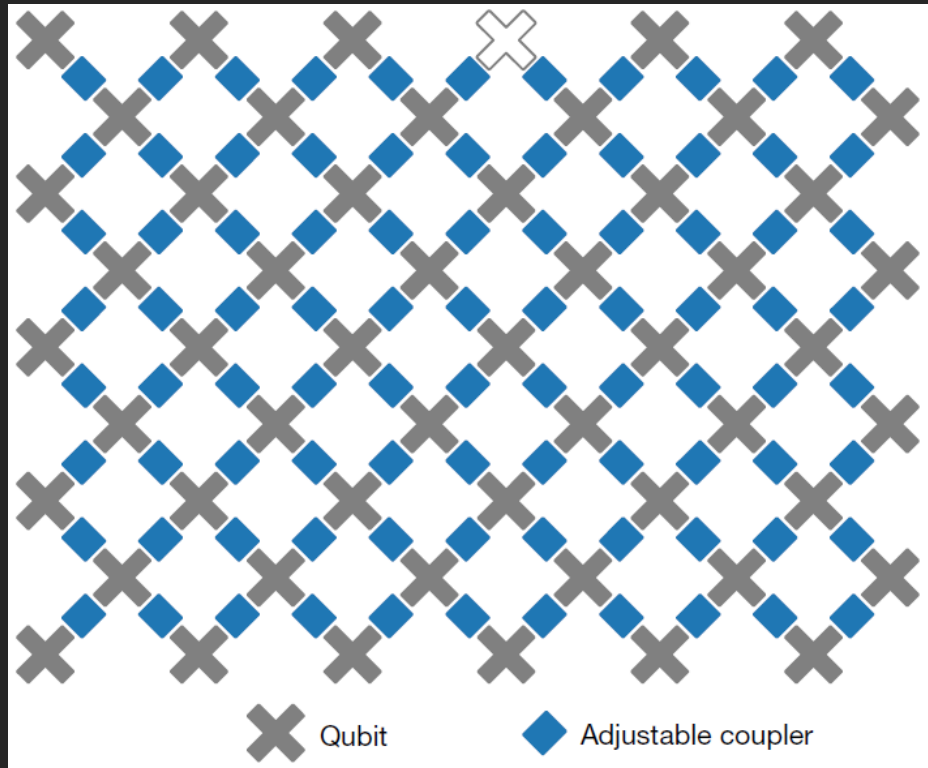
Quantum Processing Unit



Measurement Samples

Hardware Providers

Google Sycamore (54 qubits)

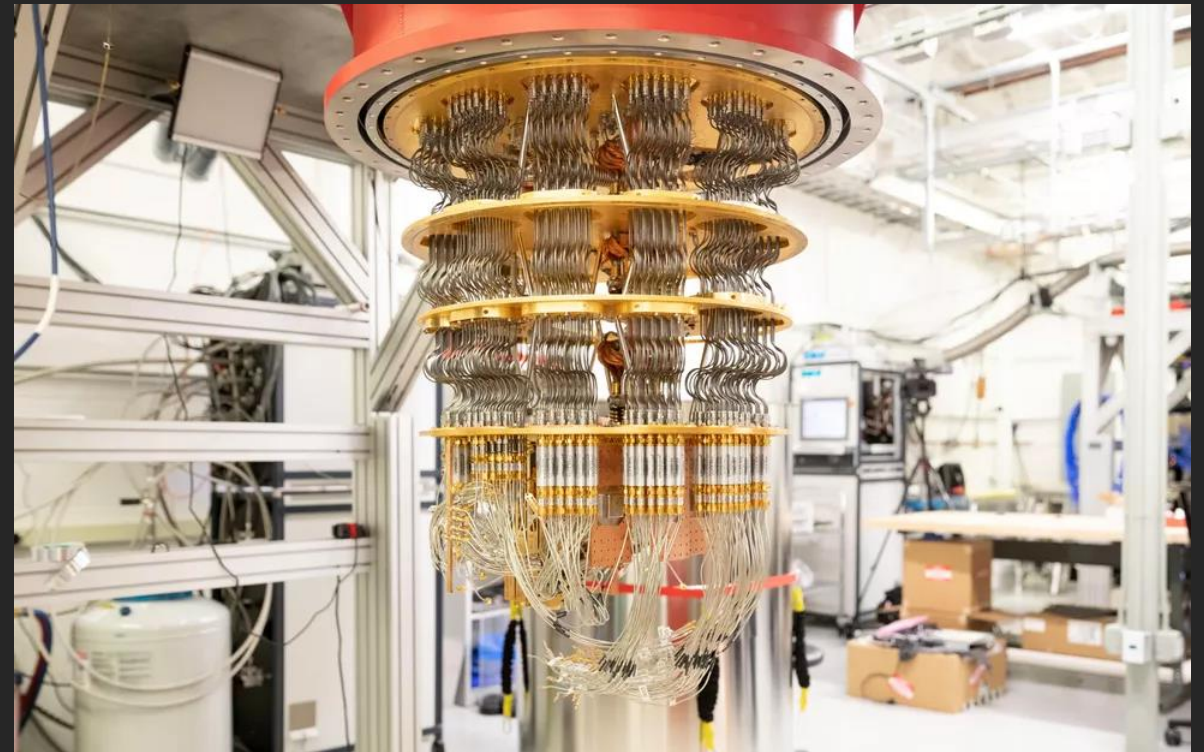
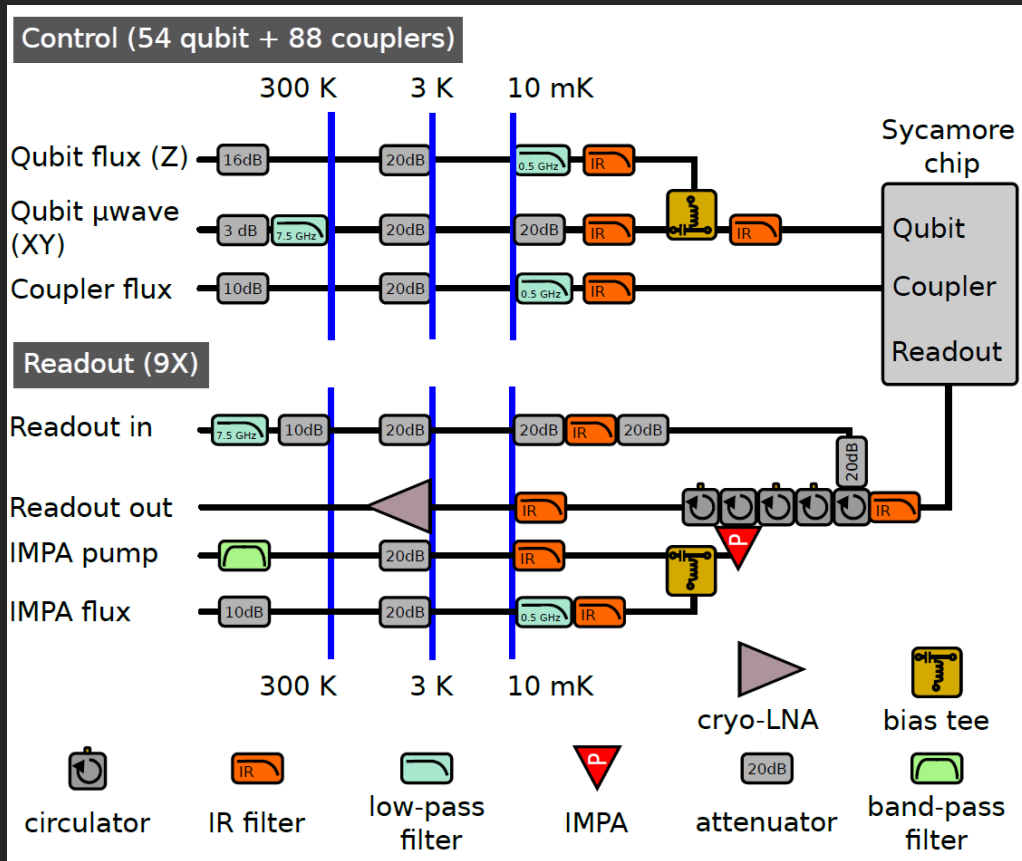


Credit: Stephen Shankland/CNET

Credit: Nature/Google

<https://doi.org/10.1038/s41586-019-1666-5>

Google Sycamore (54 qubits)

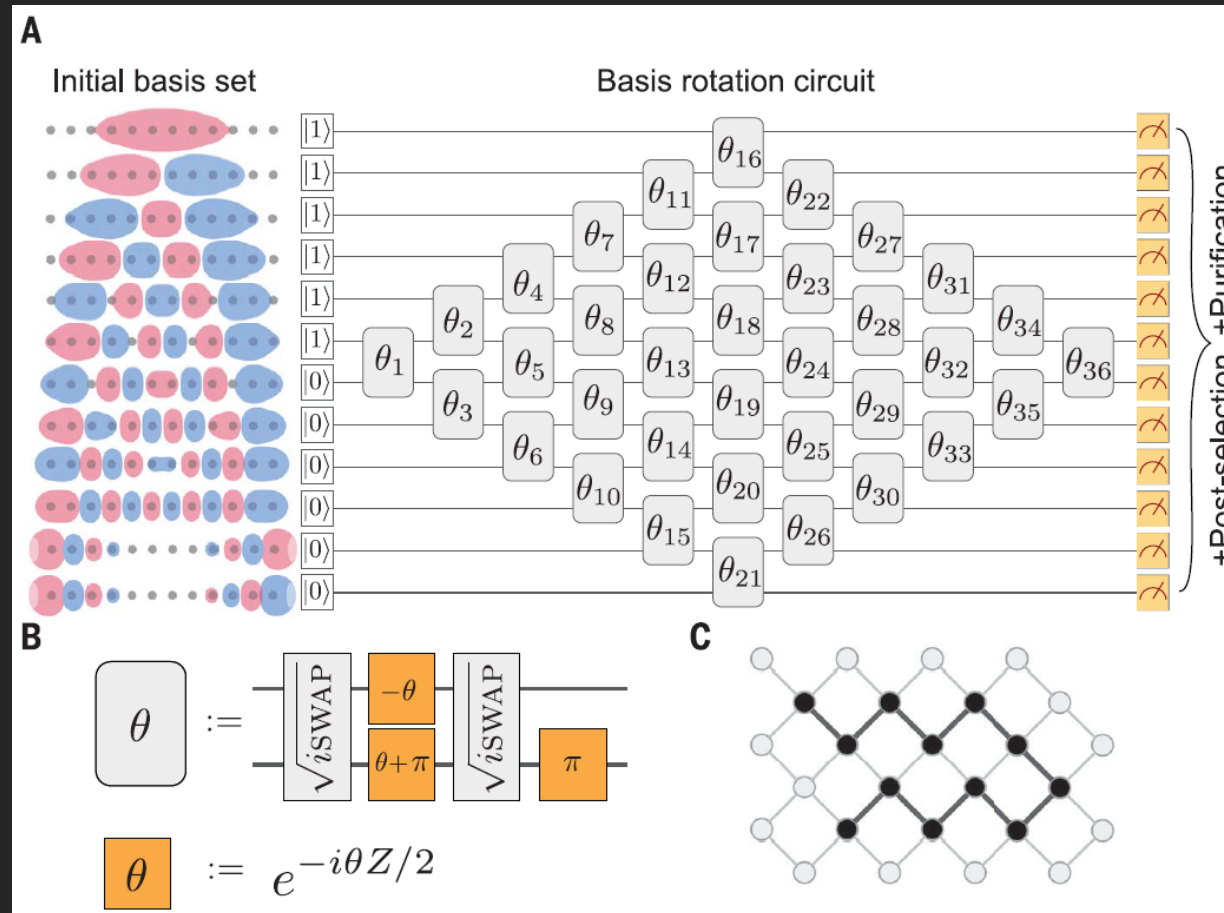


Credit: Stephen Shankland/CNET

Credit: Nature/Google

<https://doi.org/10.1038/s41586-019-1666-5>

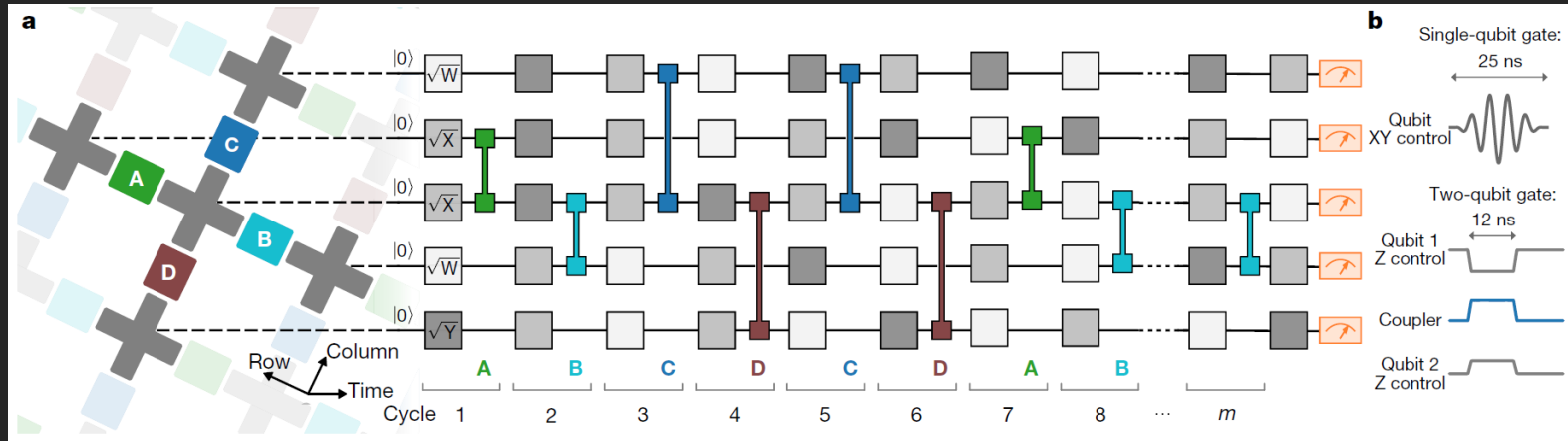
Quantum Chemistry Simulations



Credit: Science/Google

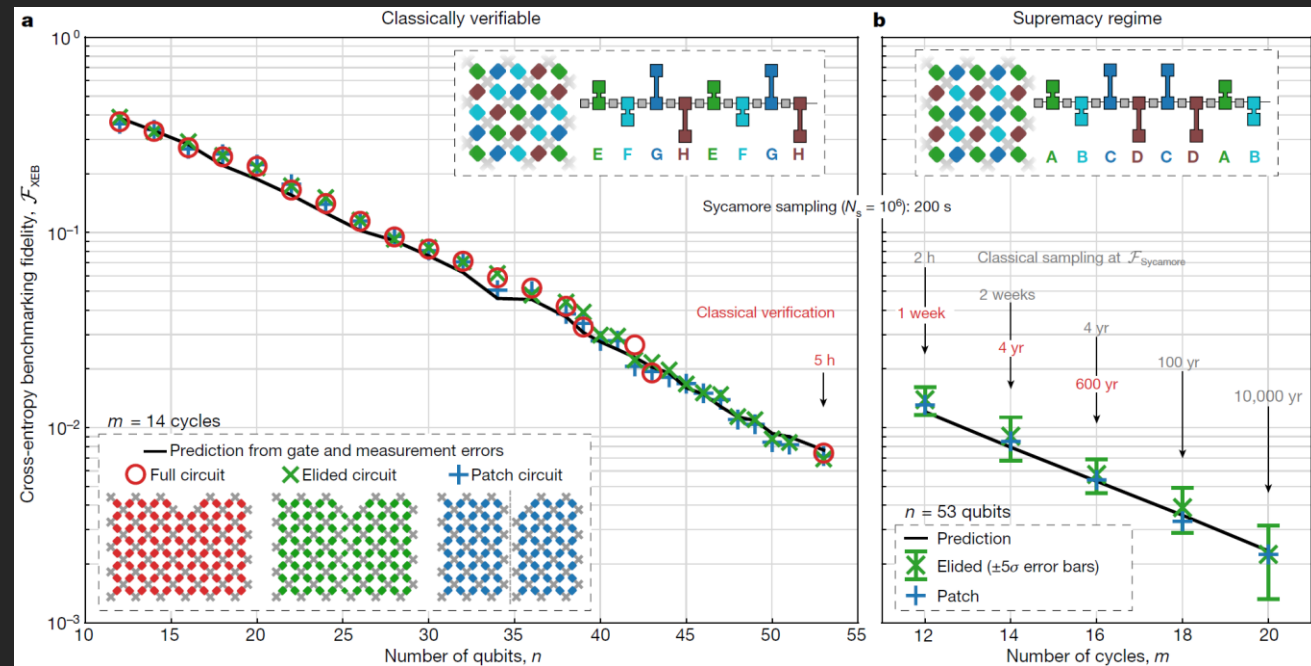
<https://doi.org/10.1126/science.abb9811>

Randomized Benchmarking (XEB)

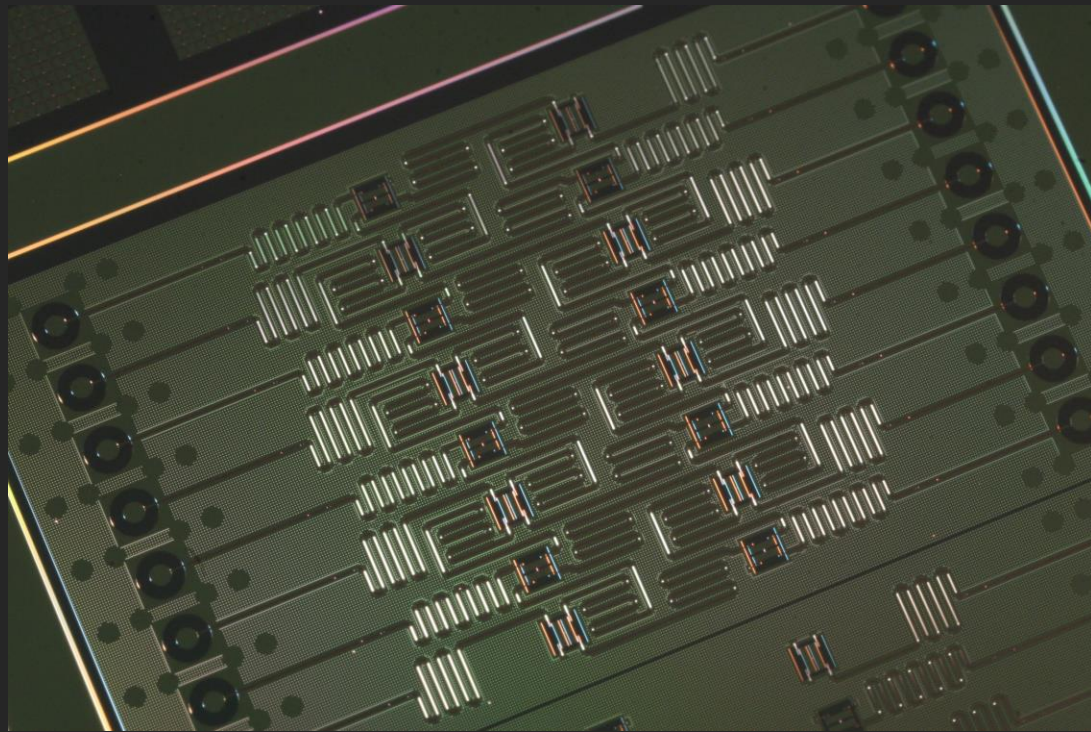


Credit: Nature/Google

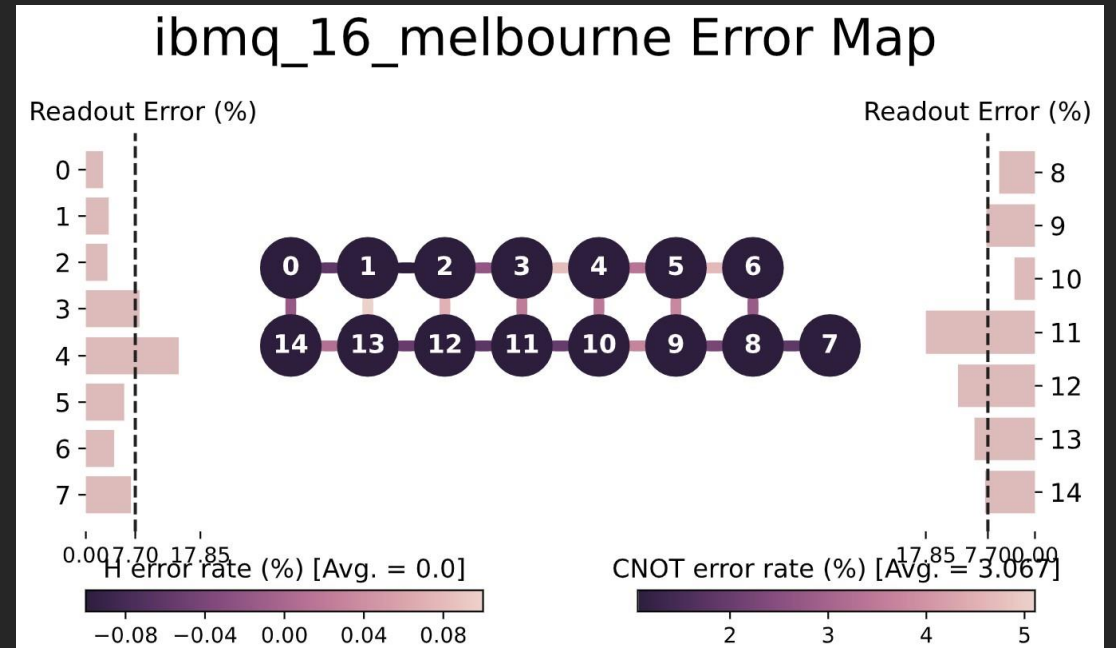
<https://doi.org/10.1038/s41586-019-1666-5>



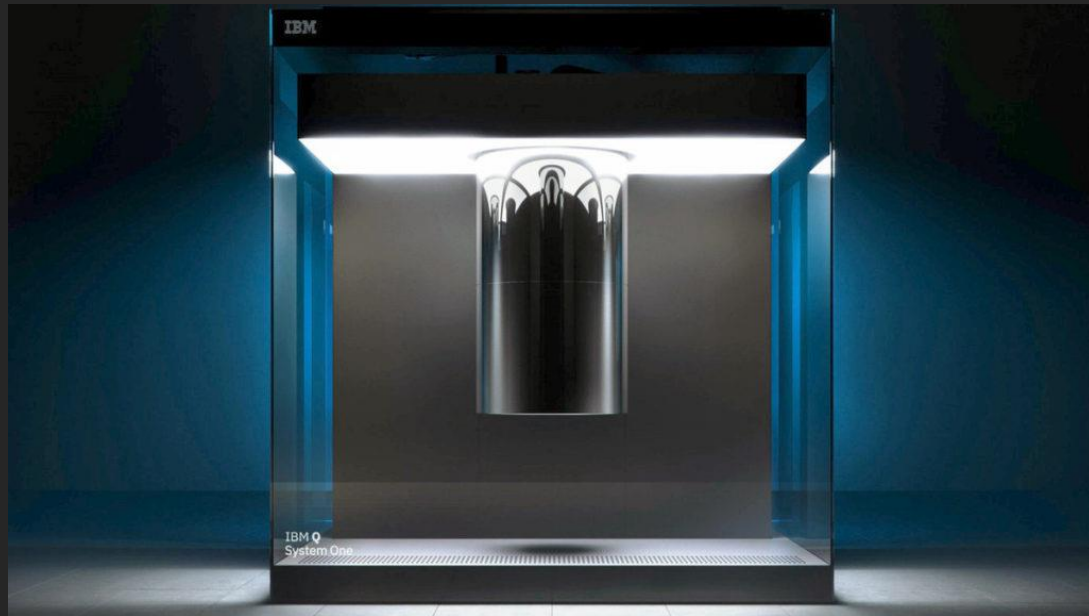
IBMQ Melbourne (16 qubits)



Credit: IBM



IBMQ System One (20 qubits)



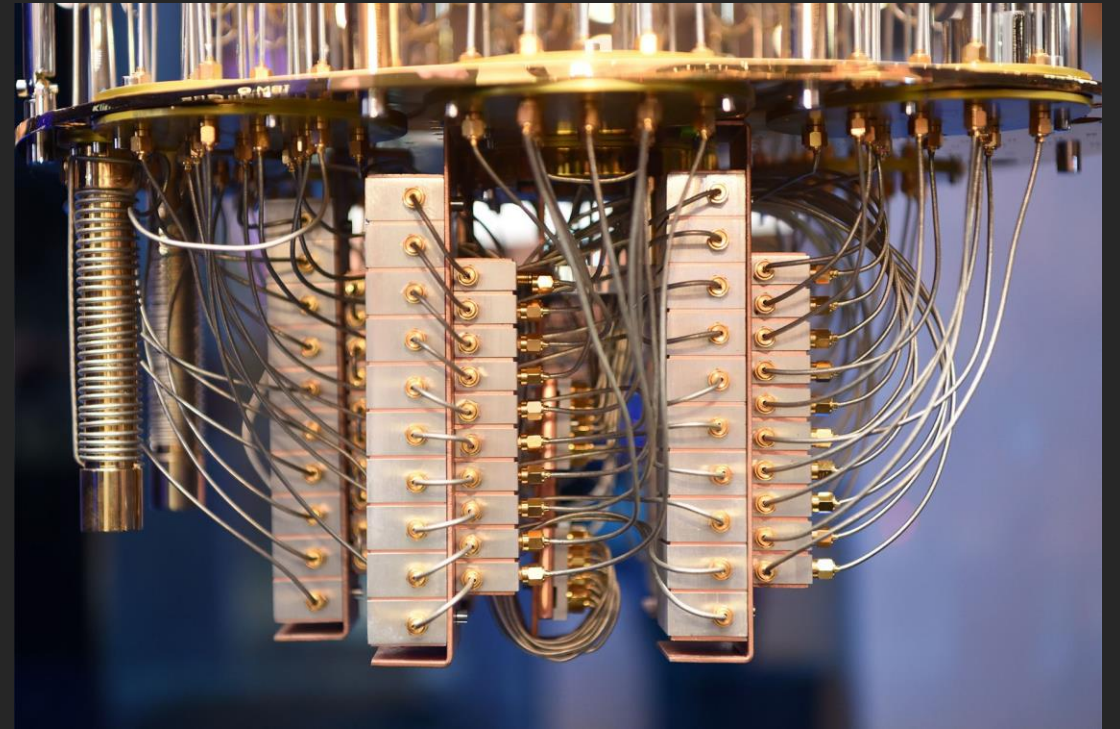
Credit: IBM



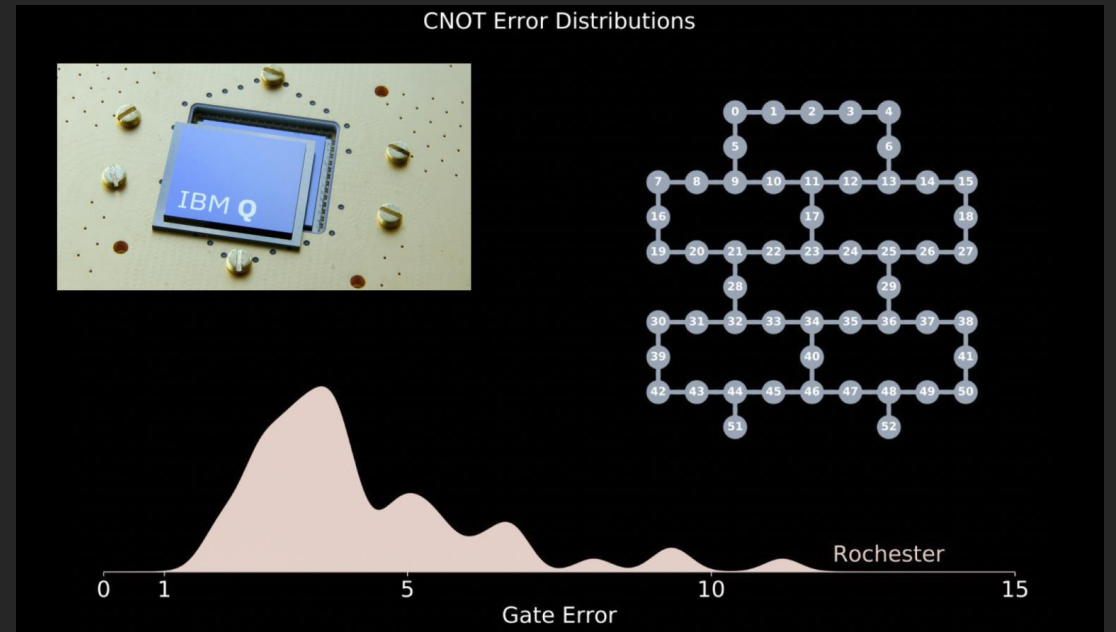
IBMQ System One (20 qubits)



Credit: IBM

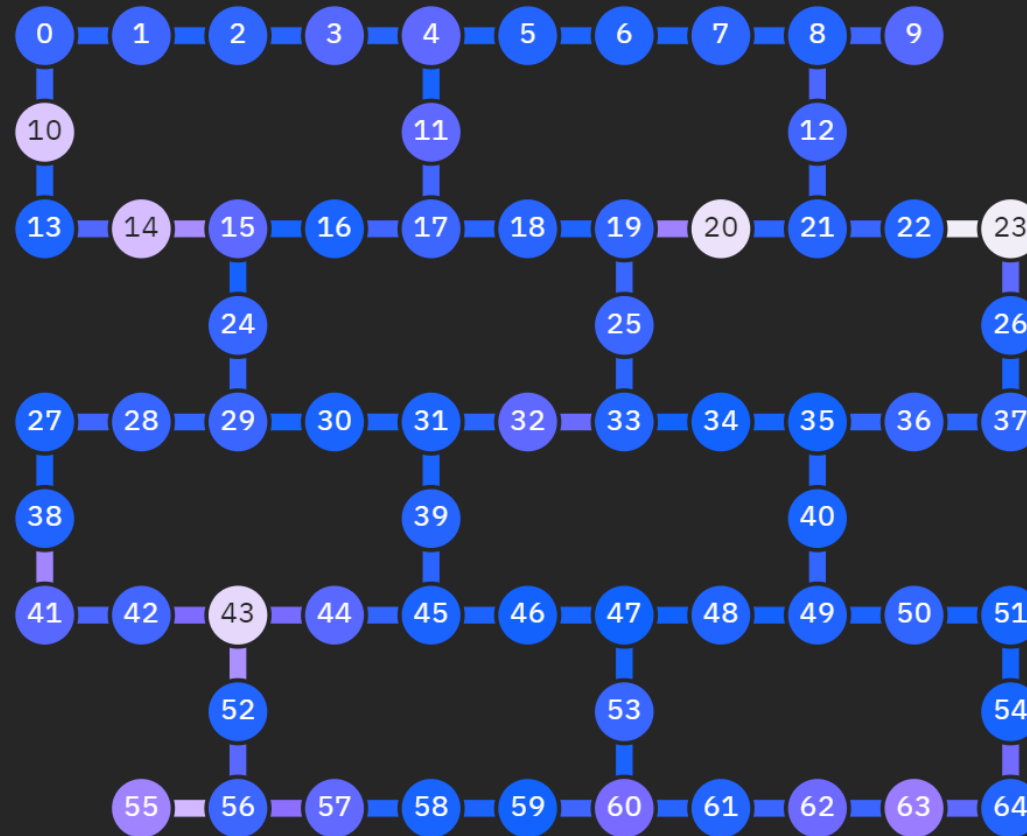


IBMQ Rochester (53 qubits)

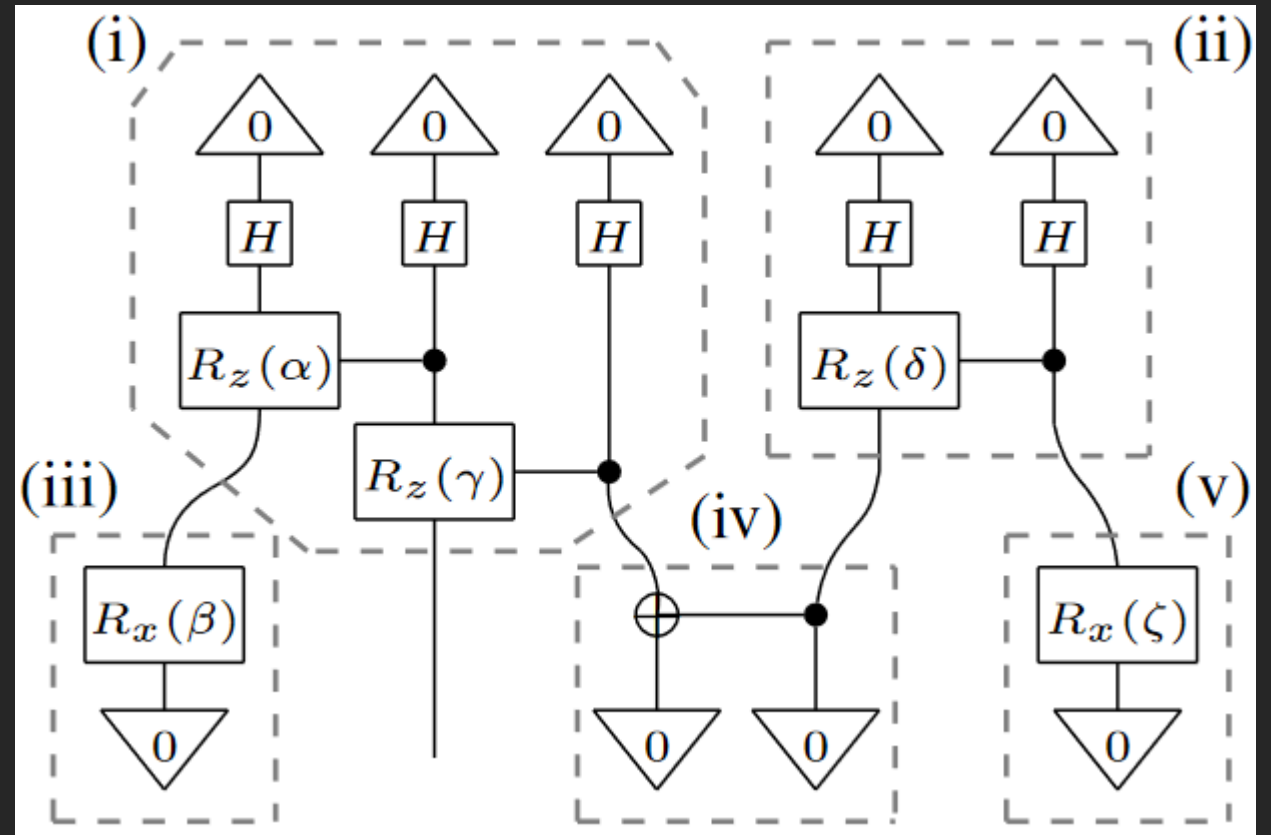
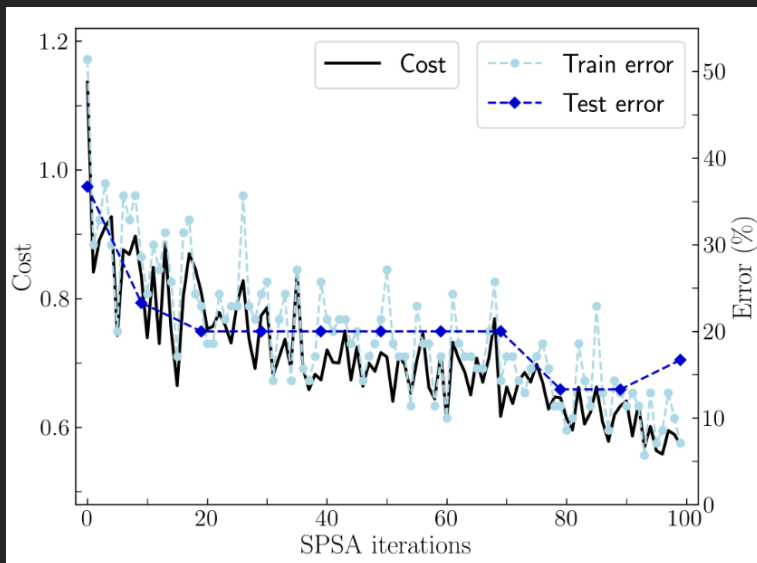
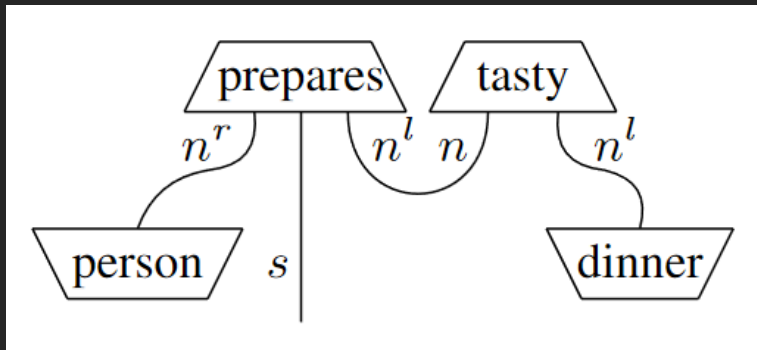


Credit: IBM

IBMQ Manhattan (65 qubits)

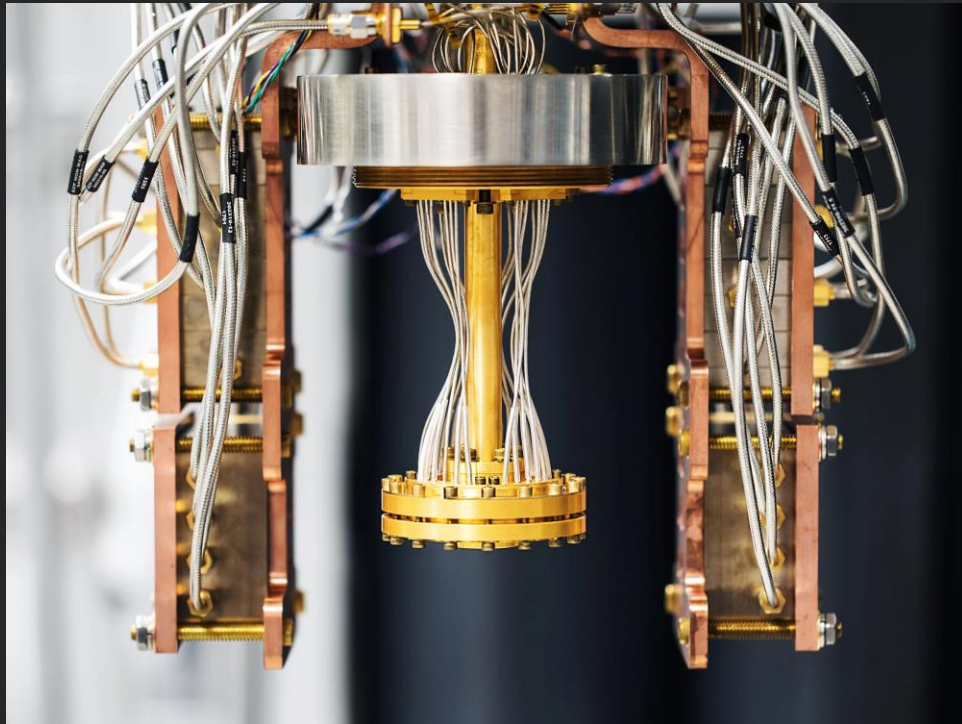


Quantum Natural Language Processing



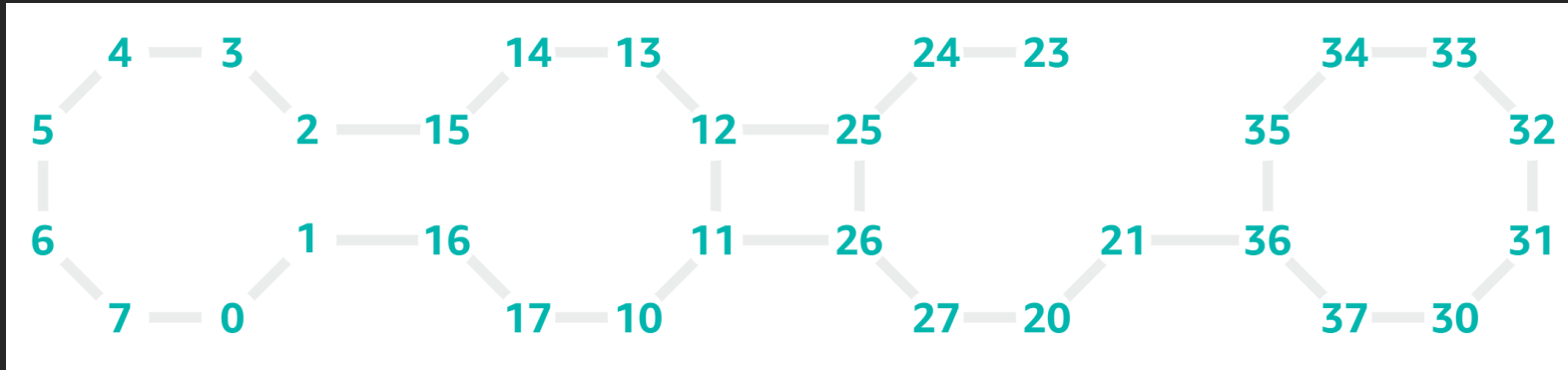
Credit: Cambridge Quantum Computing (on IBMQ)
<https://arxiv.org/abs/2102.12846>

Rigetti Acorn (19 qubits)



Credit: Rigetti

Rigetti Aspen-9 (31 qubits)



Credit: Amazon AWS/Rigetti

Credit: Rigetti

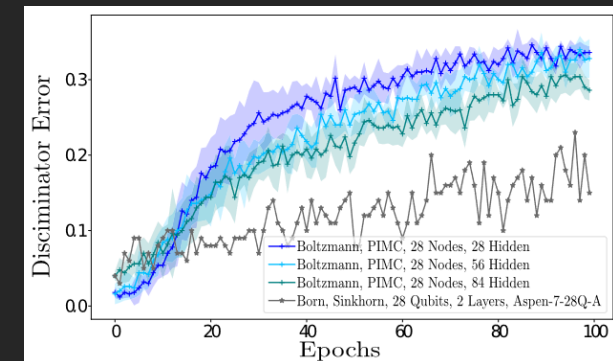
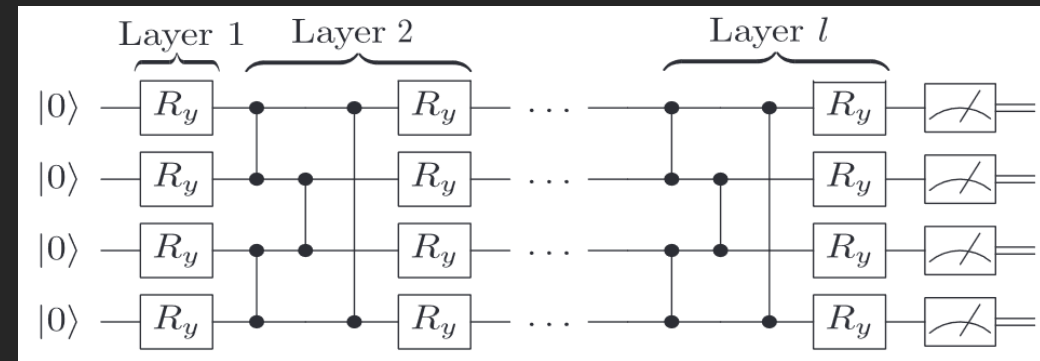
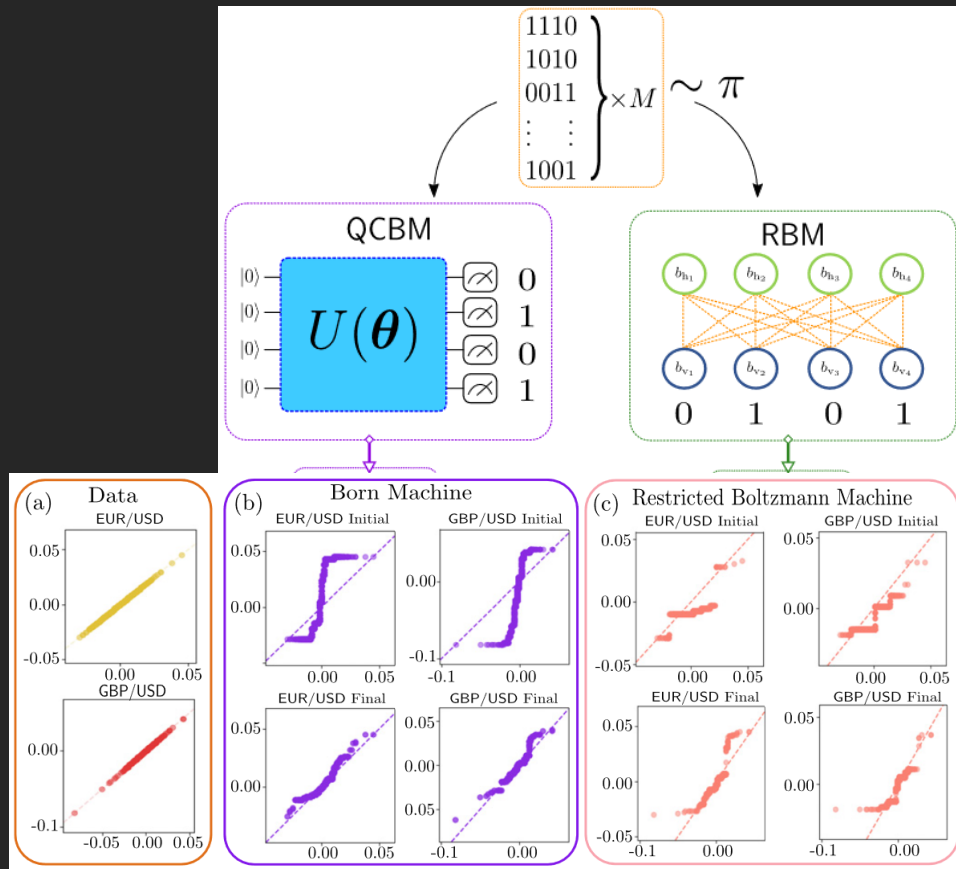
System

Architecture	Aspen
Qubits on device	32
Rep rate	50 kHz
Date deployed	May 20, 2020

Performance Snapshot

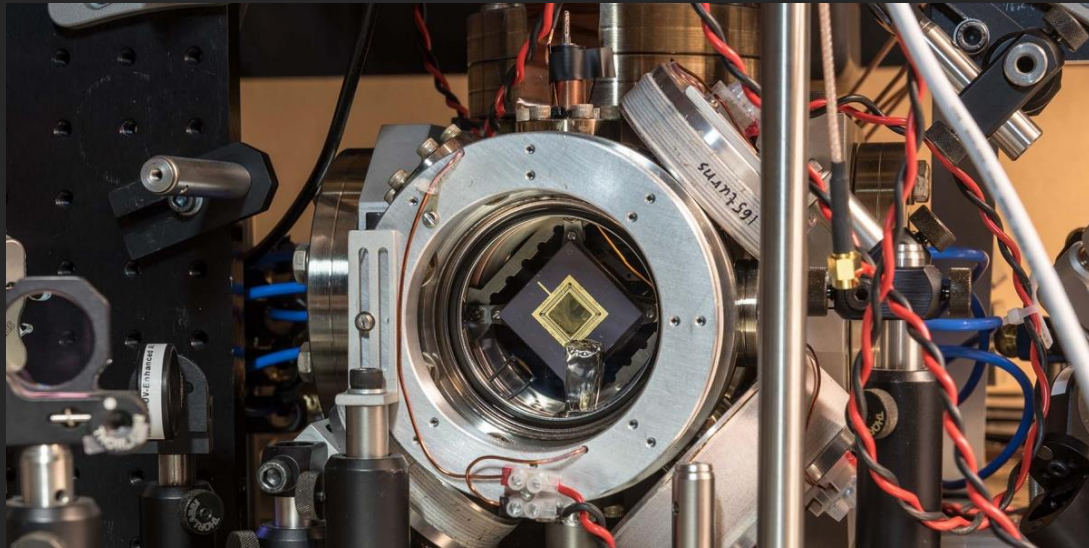
Median T1	30.1 μ s
Median T2	17.5 μ s
Median Sim 1Q Fidelity	99.3%
Median 2Q XY Fidelity	93.6%
Median 2Q CZ Fidelity	92.6%
Median RO Fidelity	95.3%
Median Active Reset Fidelity	99.4%

Generative Financial Modelling

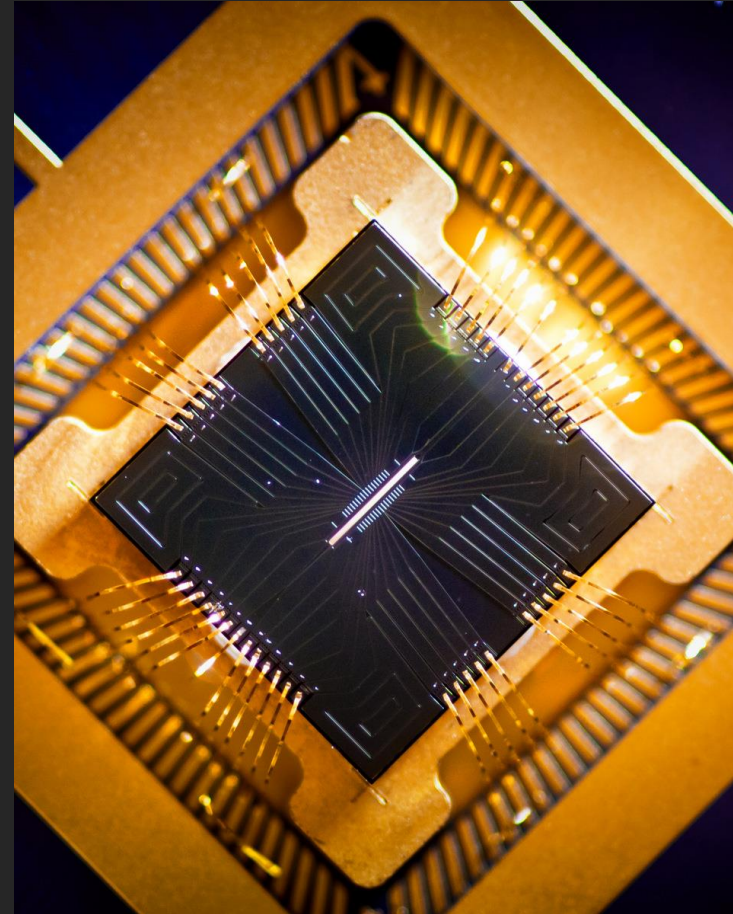


Credit: IOP/various institutions (on Rigetti HW)
<https://doi.org/10.1088/2058-9565/abd3db>

Ion-trap Quantum Computers

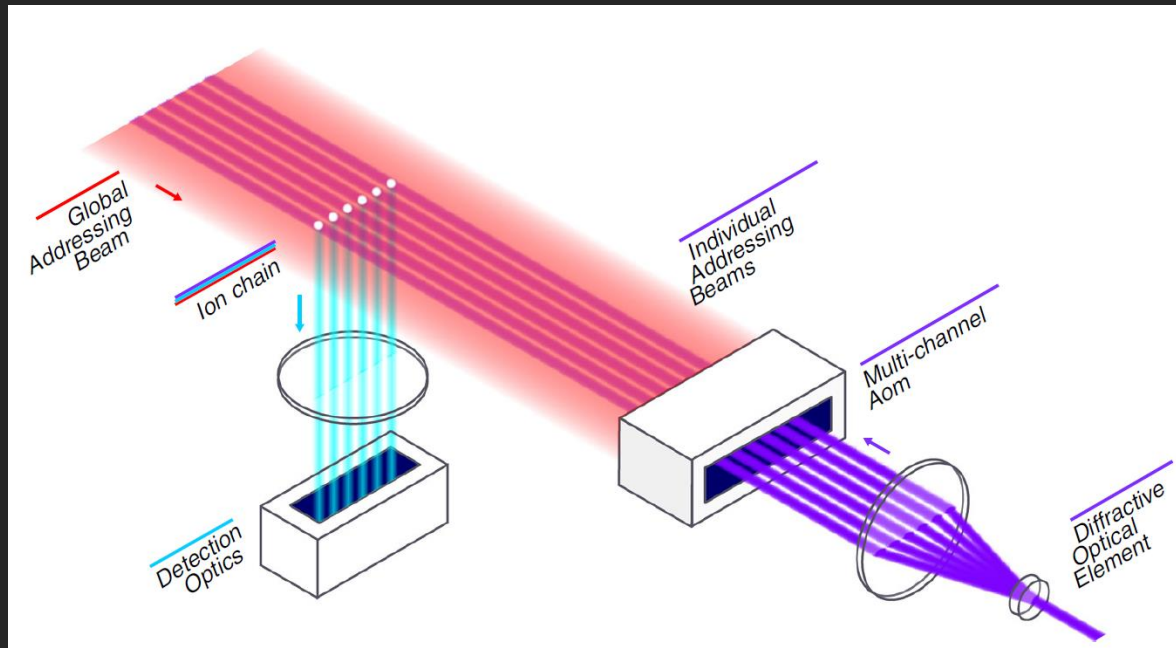


Credit: NQIT/Stuart Bebb



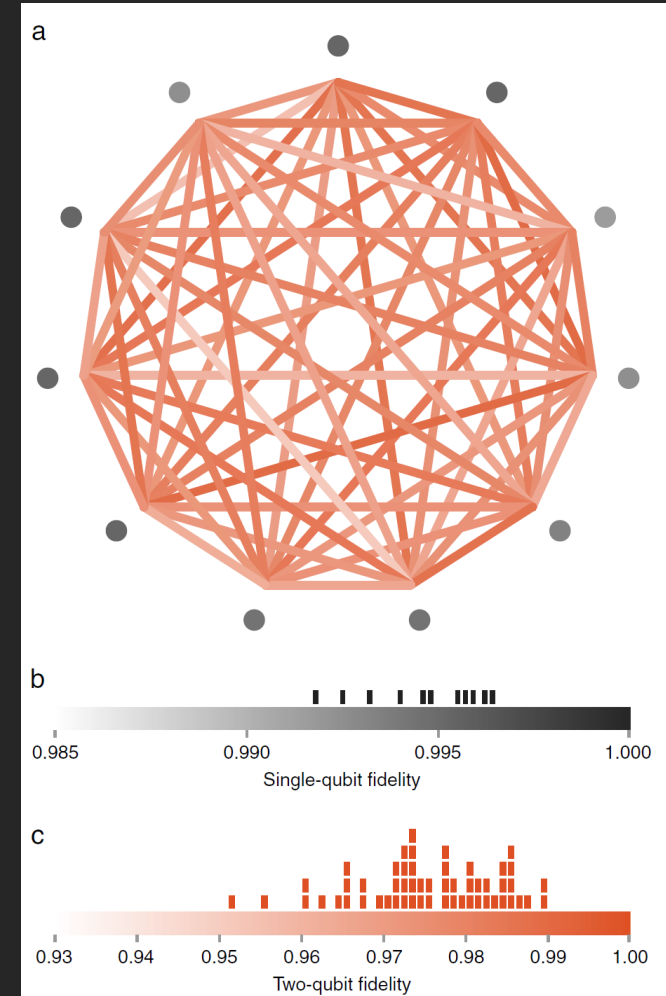
Credit: JQI

IonQ 171Yb+ (11 qubits)

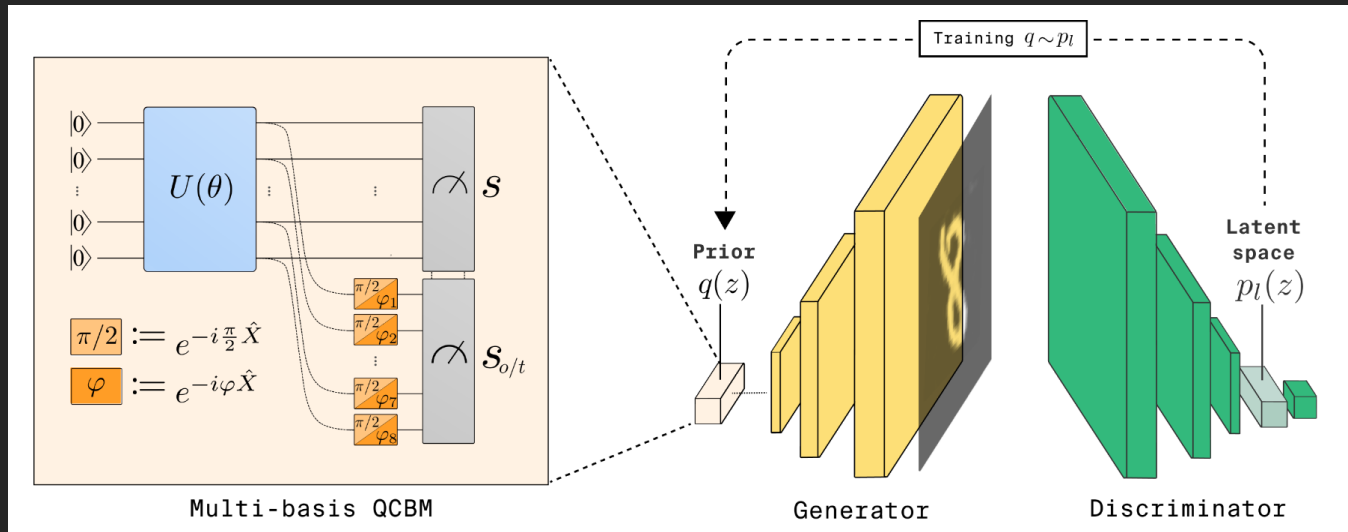


Credit: Nature/IonQ

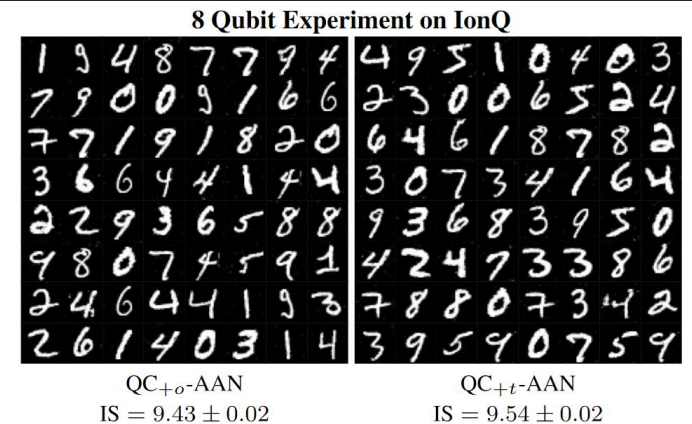
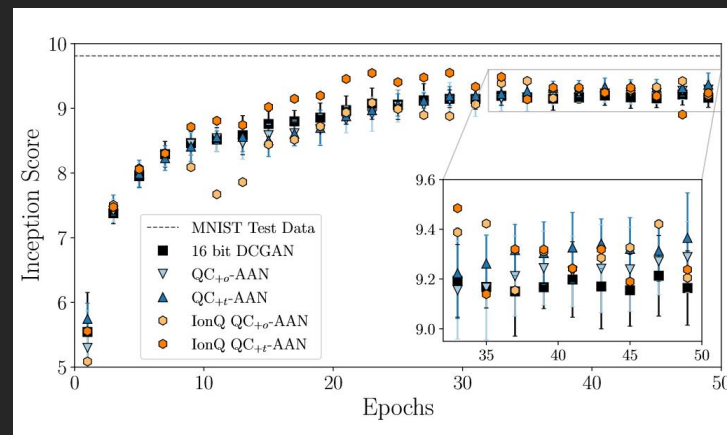
<https://doi.org/10.1038/s41467-019-13534-2>



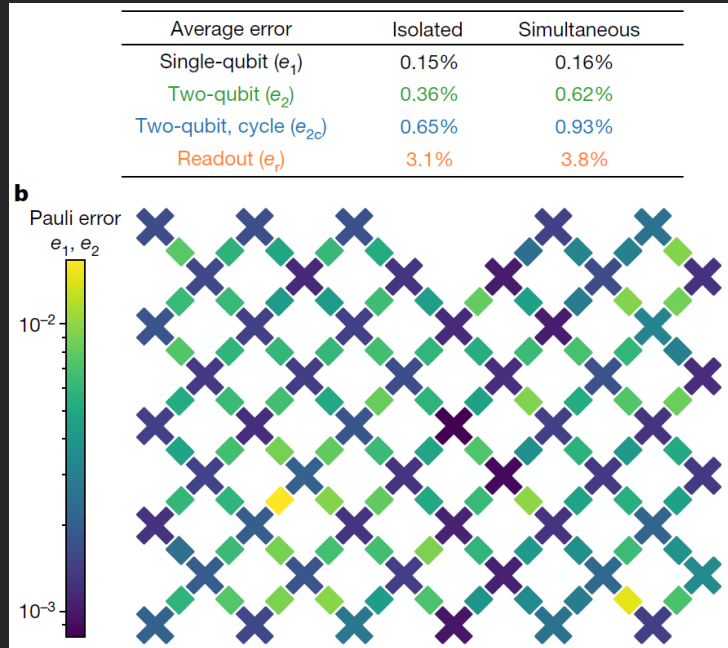
Quantum Generative Adversarial Networks



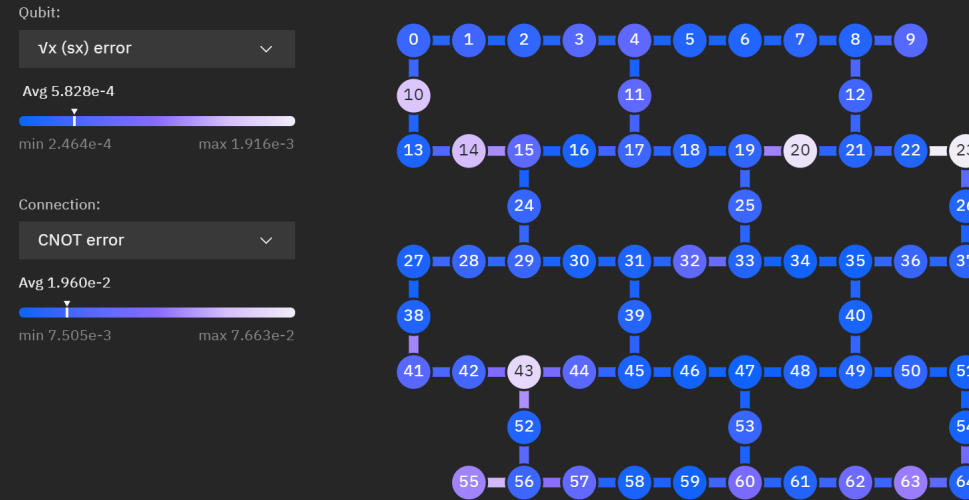
Credit: Zapata Computing/IonQ
<https://arxiv.org/abs/2012.03924>



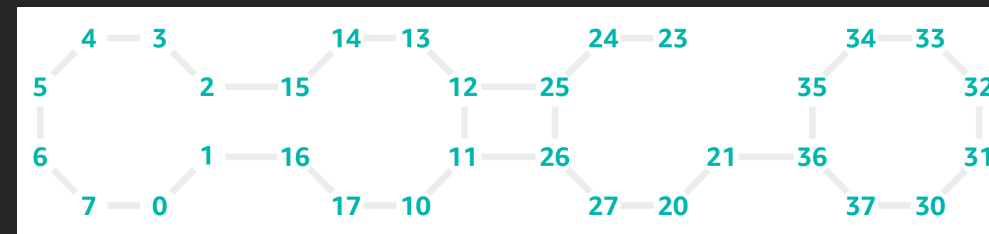
Superconducting vs Ion-trap



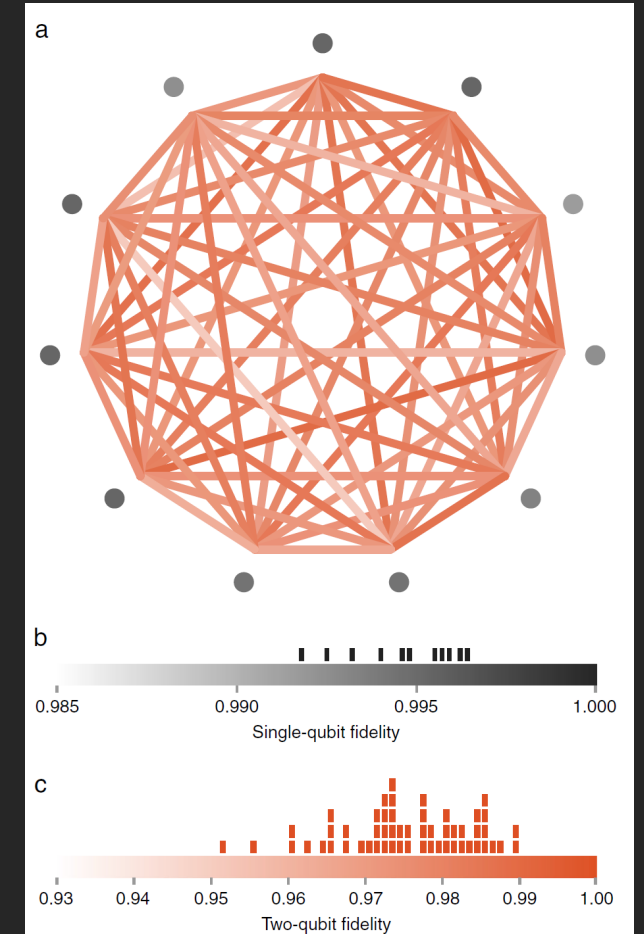
Credit: Nature/Google
<https://doi.org/10.1038/s41586-019-1666-5>



Credit: IBMQ

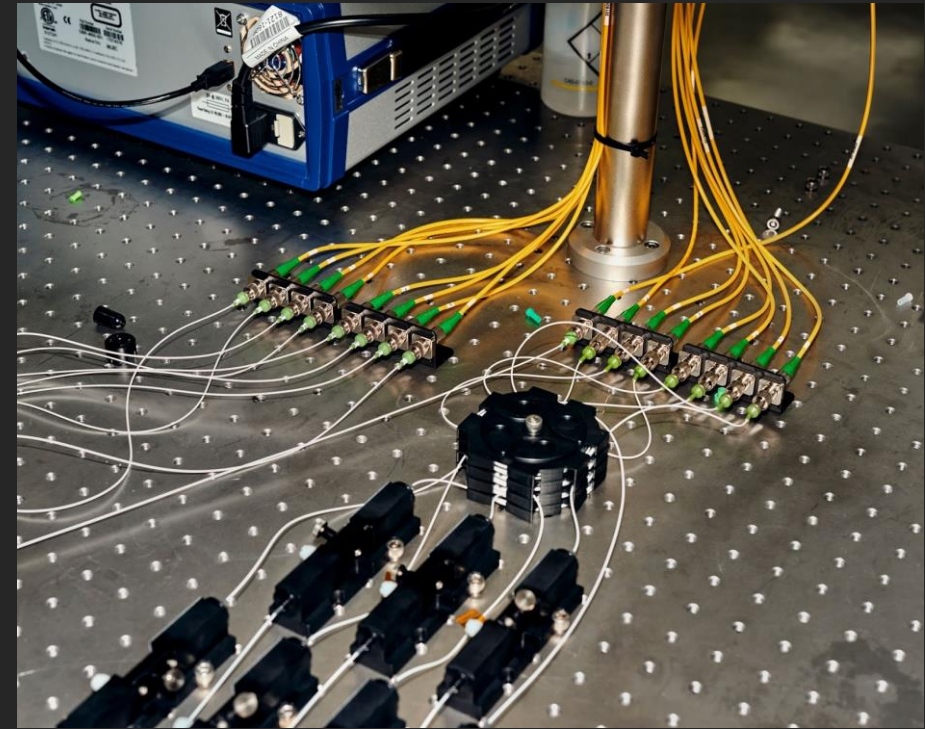
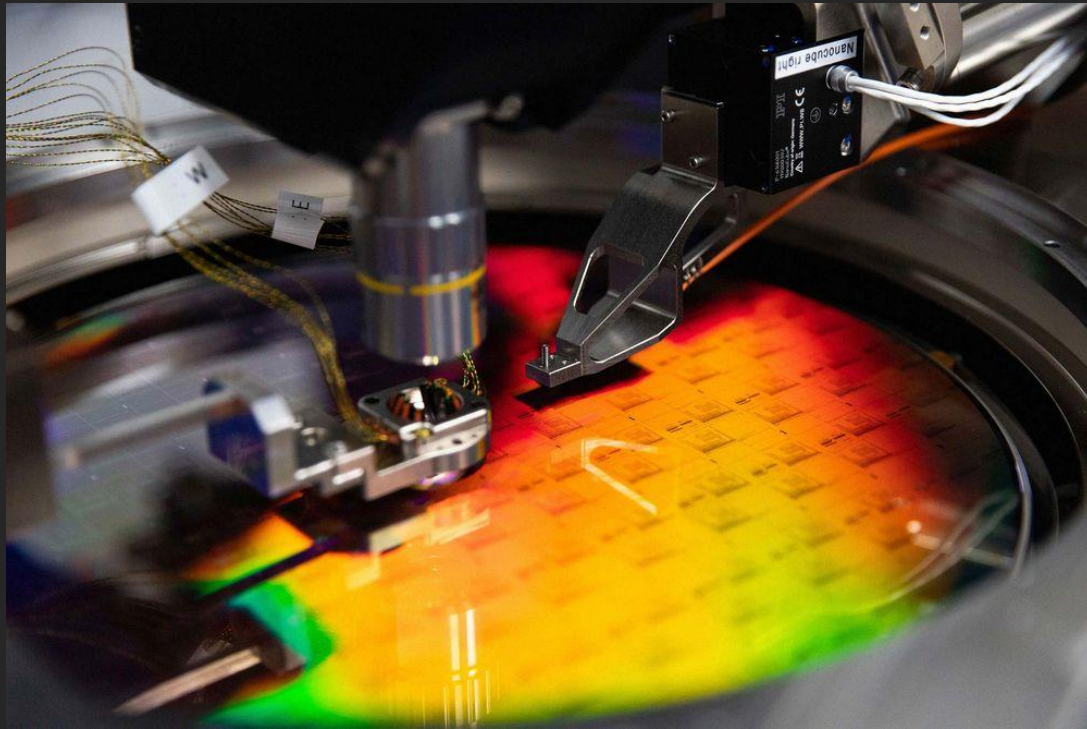


Credit: Amazon AWS/Rigetti



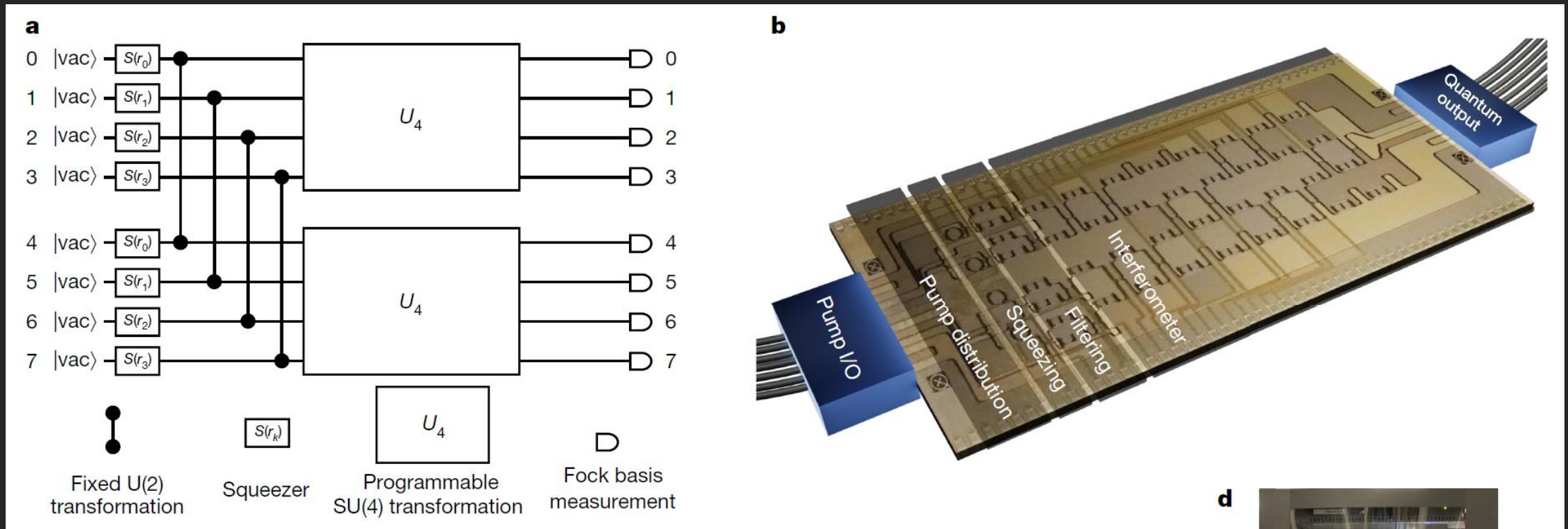
Credit: Nature/IonQ
<https://doi.org/10.1038/s41467-019-13534-2>

Photonic Quantum Computers



Credit: PsiQuantum

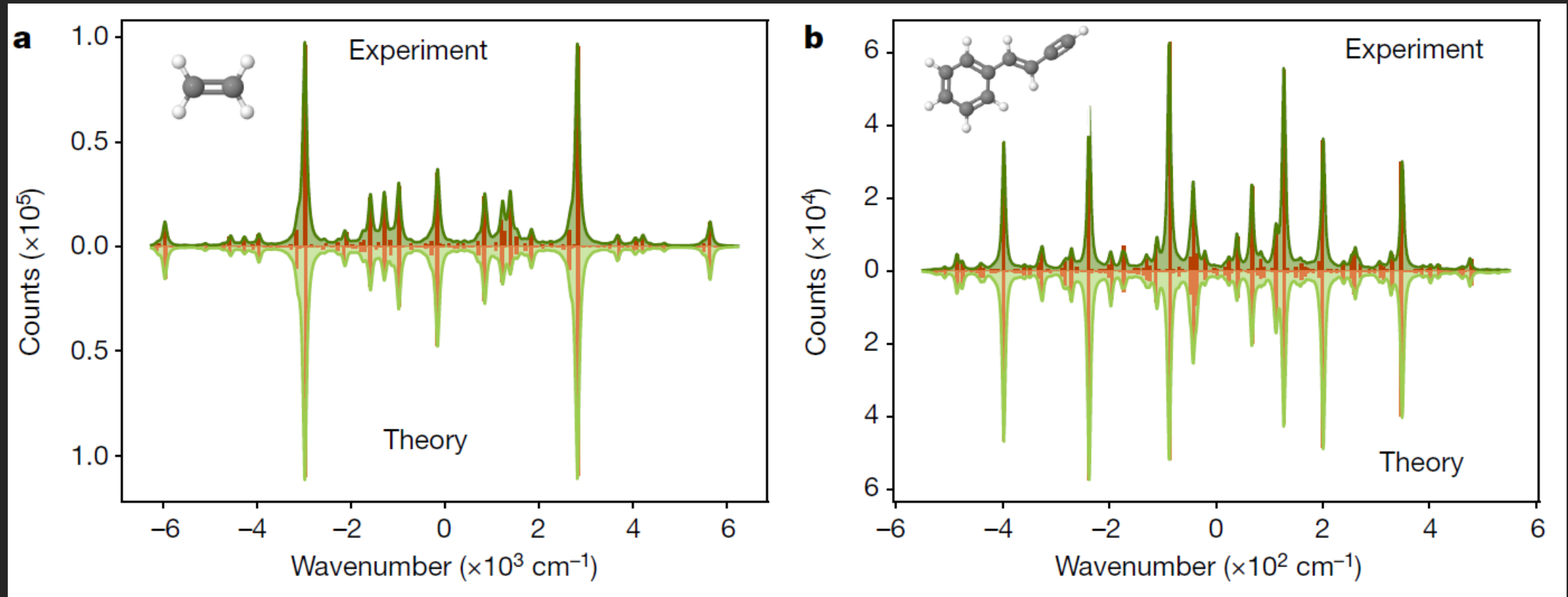
Xanadu 8-mode photonic chip



Credit: Nature/Xanadu

<https://doi.org/10.1038/s41586-021-03202-1>

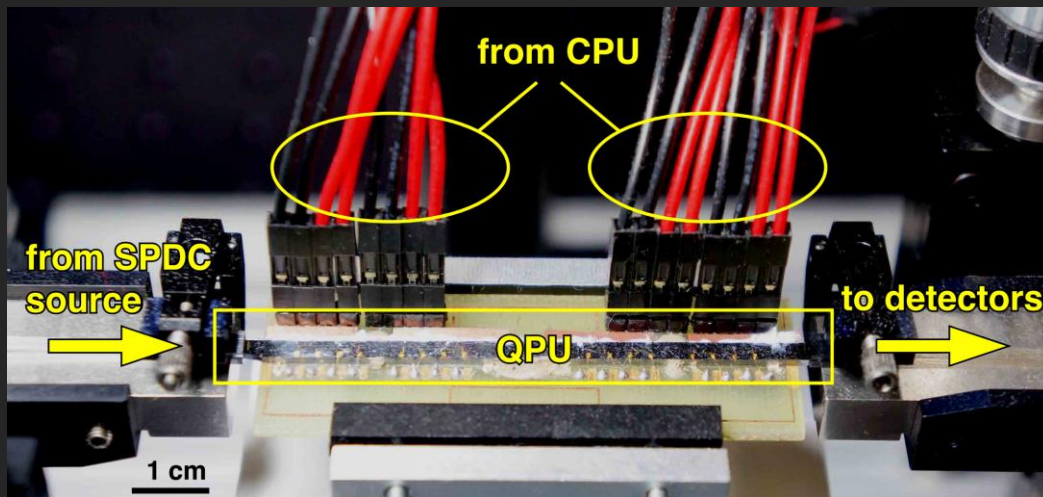
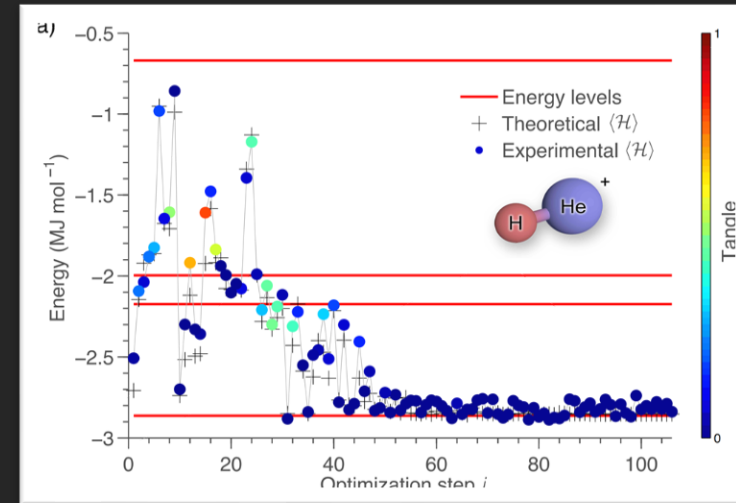
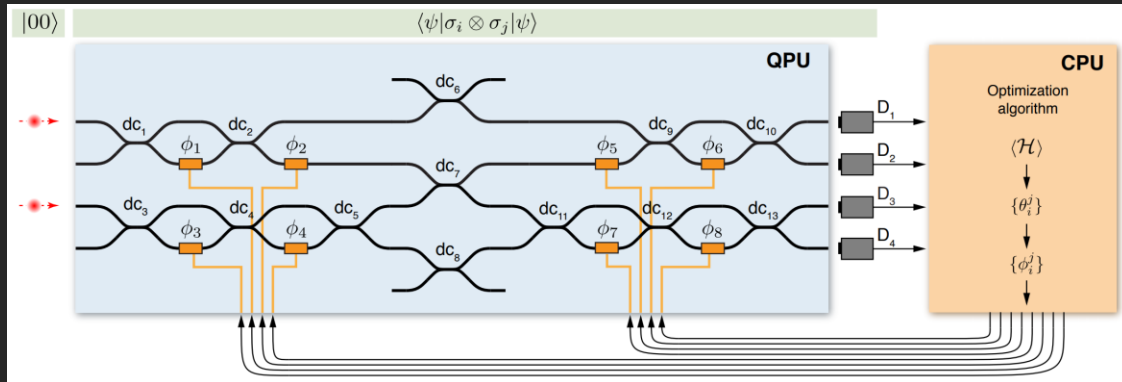
Quantum Chemistry Simulations



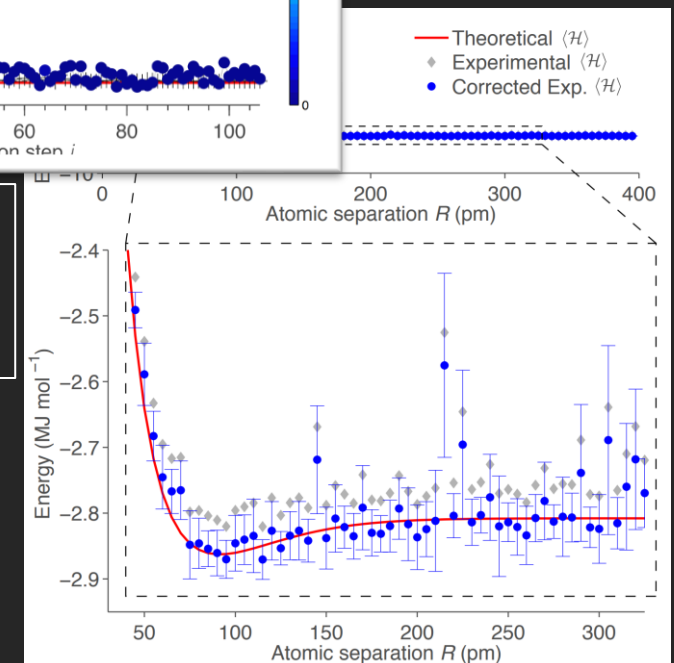
Credit: Nature/Xanadu

<https://doi.org/10.1038/s41586-021-03202-1>

Quantum Chemistry Simulations



This is a photonic QPU running VQE in 2003... Ancient QC history.

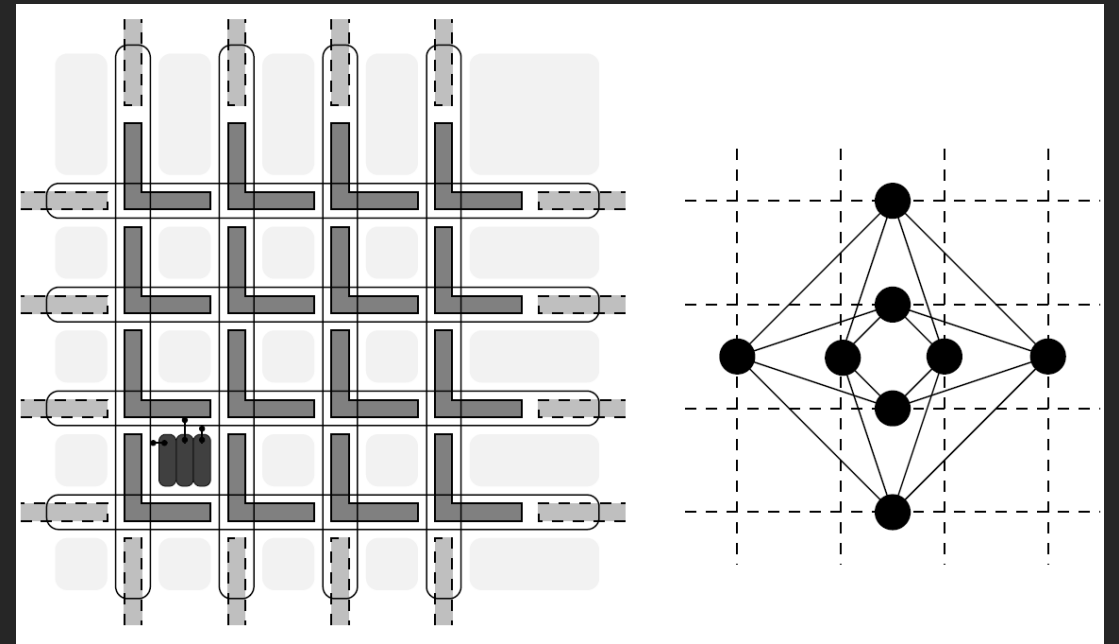
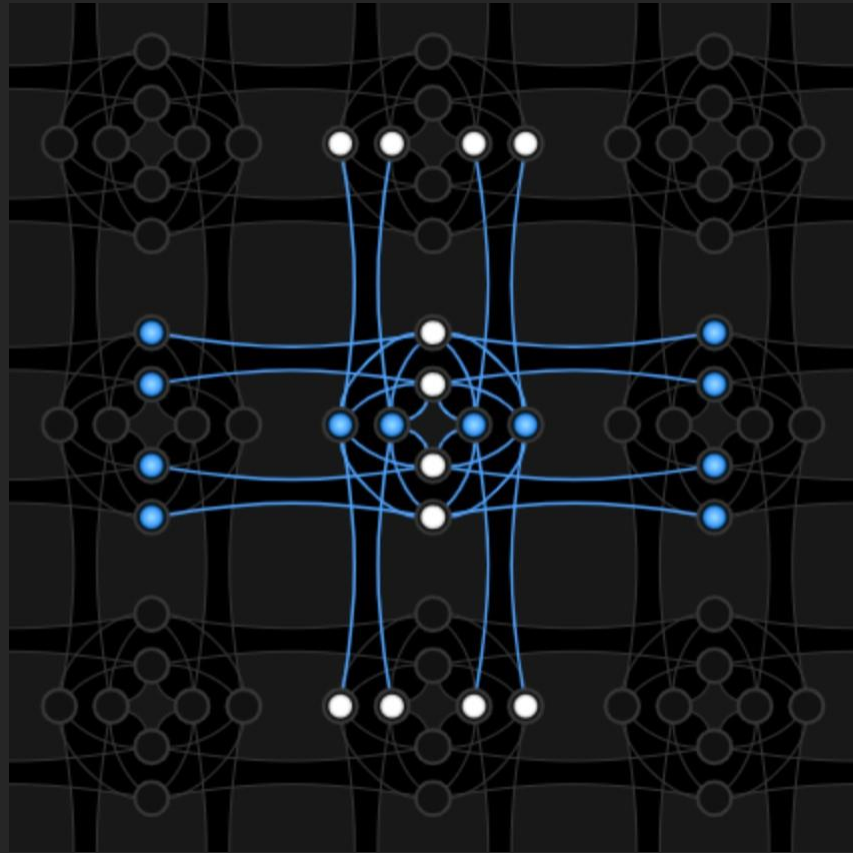


Quantum Annealers



Credit: D-Wave

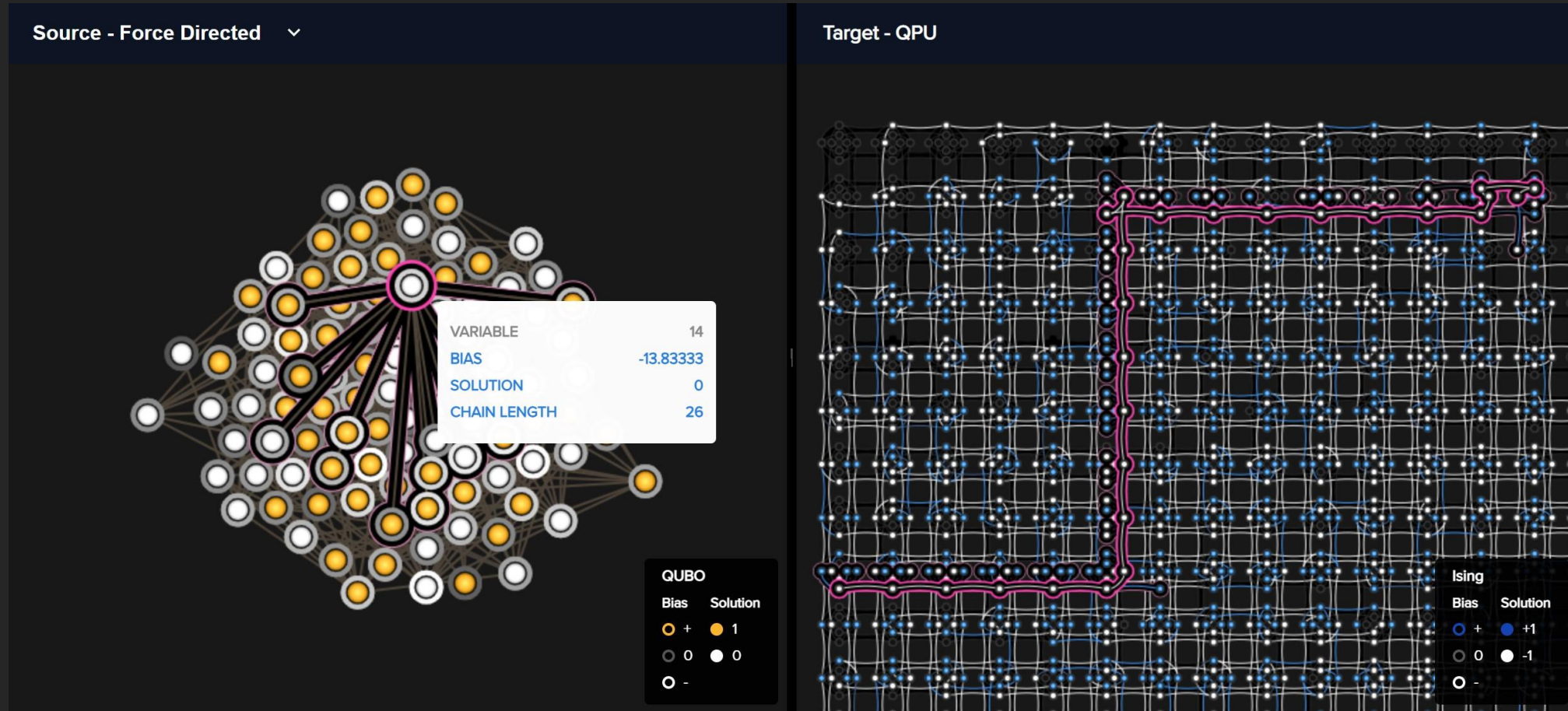
D-Wave 2000Q (2048 qubits, Chimera topology)



<https://doi.org/10.1109/TASC.2014.2318294>

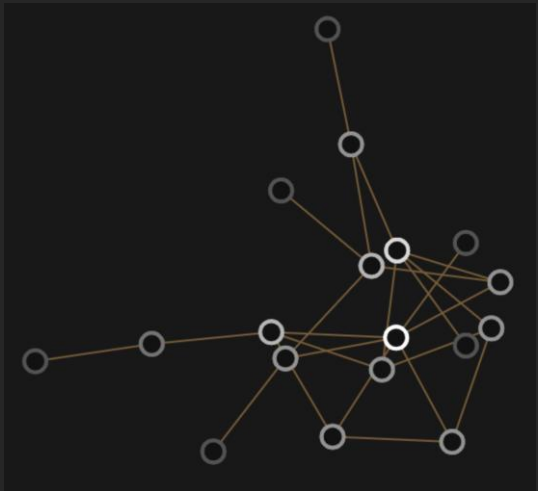
Credit: D-Wave

Quantum Annealers

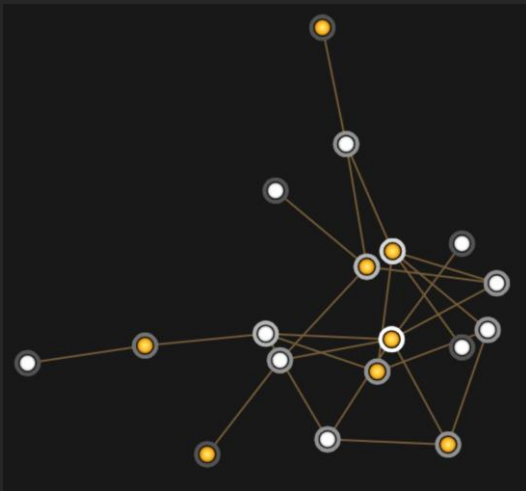


Credit: D-Wave

Example: Max-cut with QA

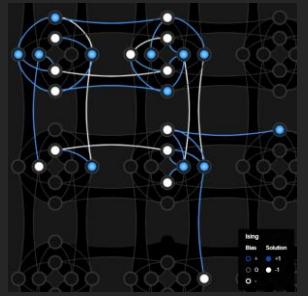


Graph

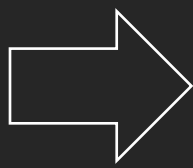
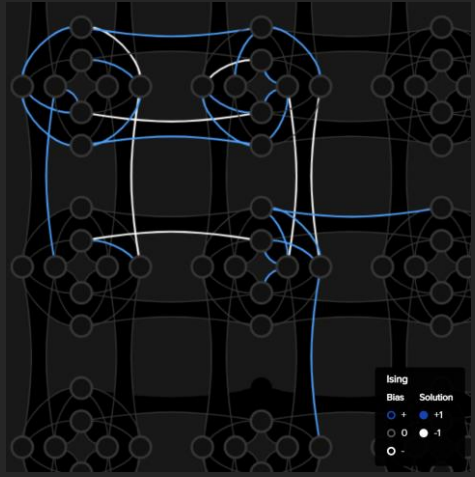


Graph Cut

Min energy solution

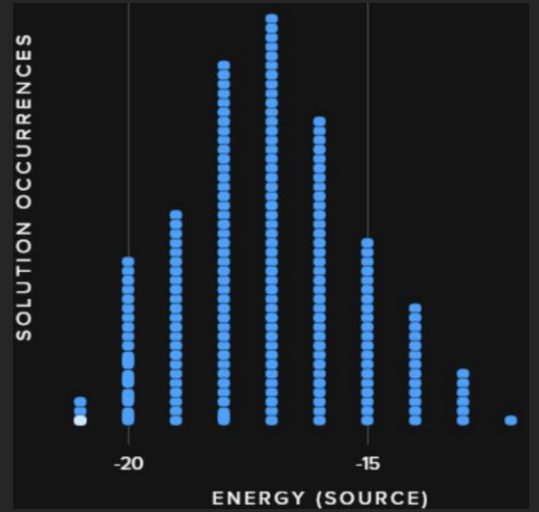
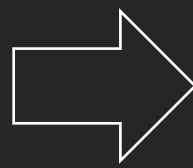


Encoding as Ising model



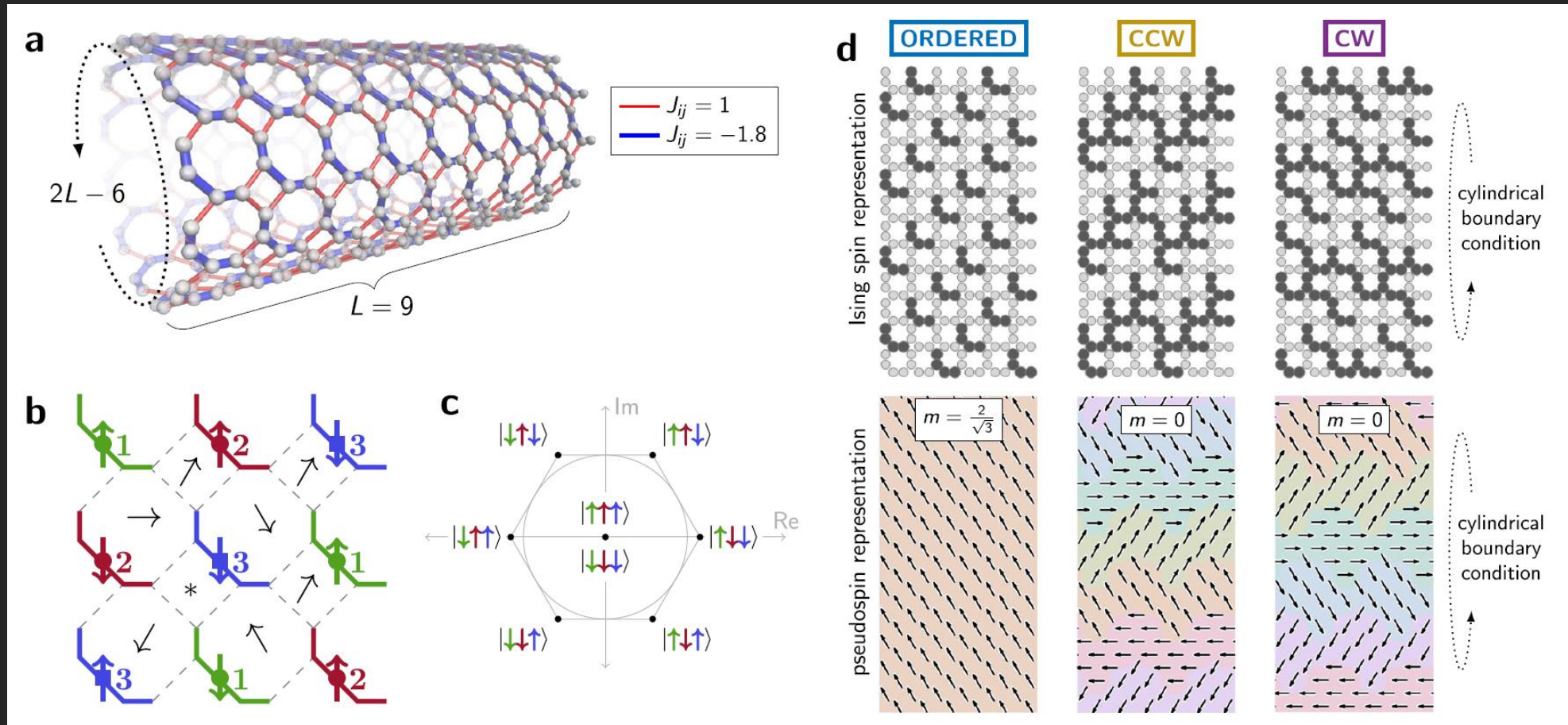
NAME (CHIP ID) DW_2000Q_6	DESCRIPTION D-Wave 2000Q lower-noise system
QUBITS 2048	SUPPORTED PROBLEM TYPES ising, qubo
TOPOLOGY [16,16,4] chimera	TAGS lower_noise

QPU



Samples

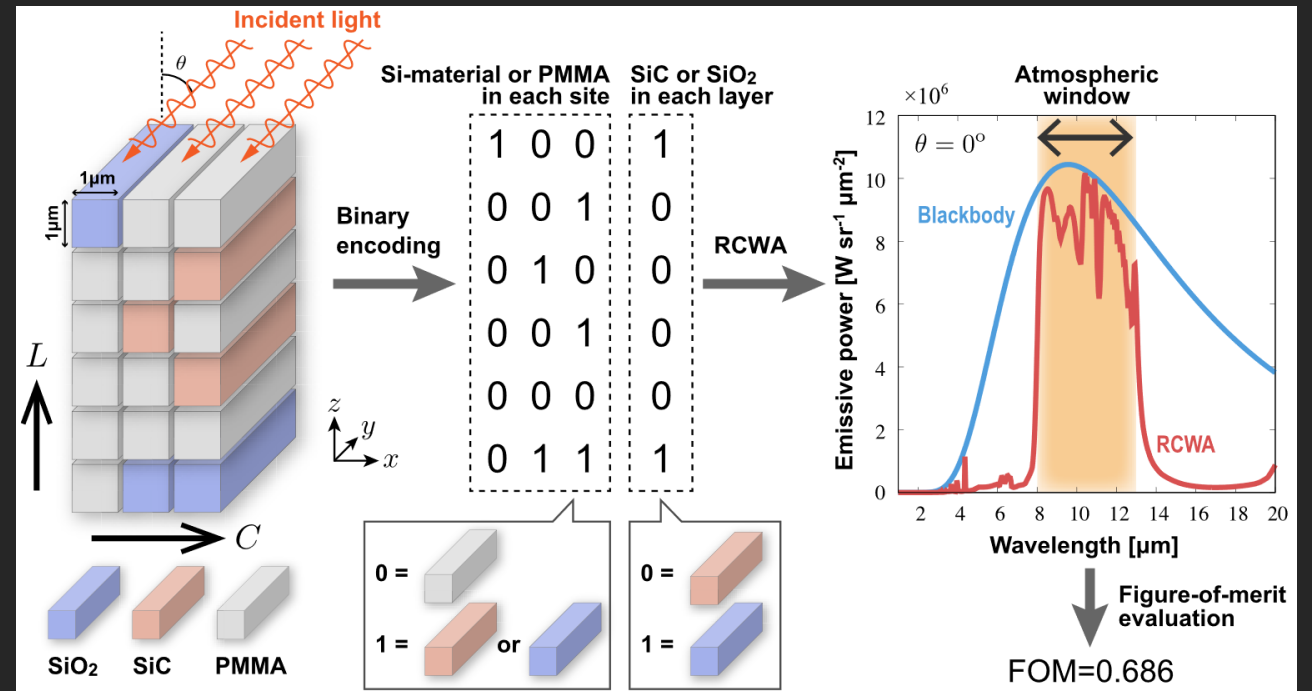
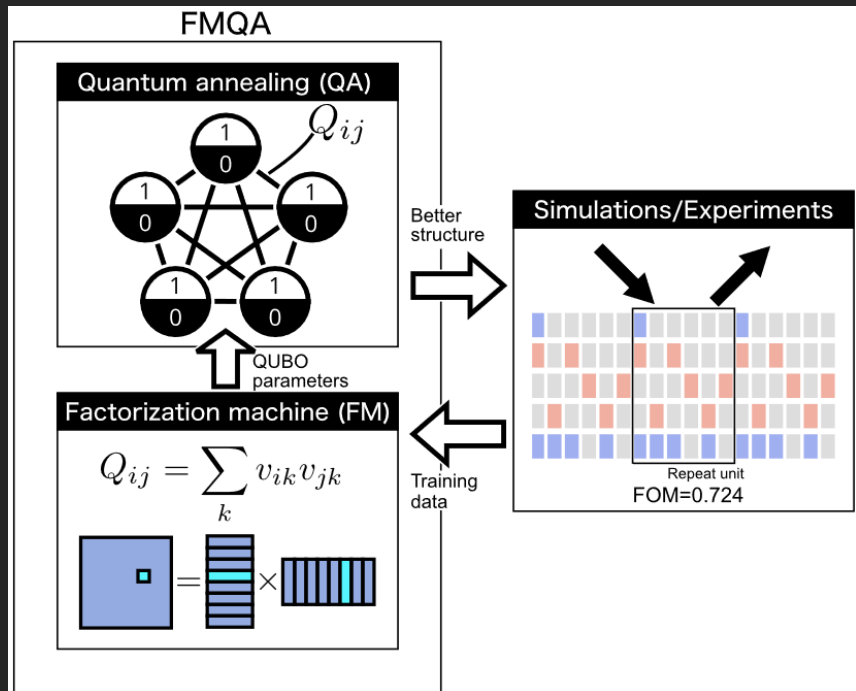
Spin-glass Simulations



Credit: Nature/D-Wave

<https://doi.org/10.1038/s41467-021-20901-5>

Metamaterial Design



Credit: APS/various institutions (on D-Wave HW)
<https://doi.org/10.1103/PhysRevResearch.2.013319>

QKD and QRNG (not QC)



Credit: ID Quantique

Software Providers

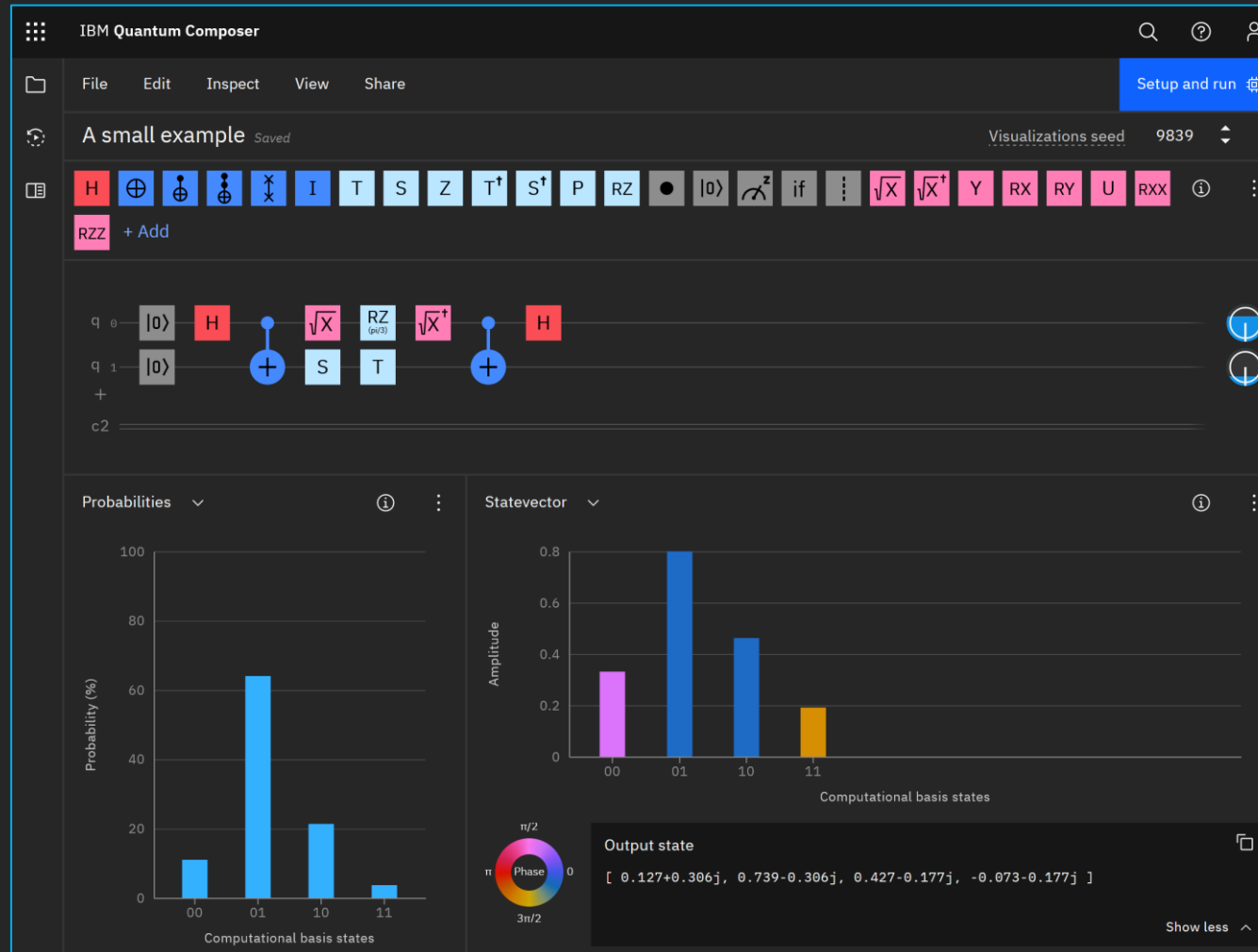
IBM Quantum

The screenshot displays the IBM Quantum Services dashboard. At the top, there's a navigation bar with 'IBM Quantum Services' and user icons. Below it, a 'Services' section provides an overview and navigation tabs for 'Programs', 'Systems', and 'Simulators'. A descriptive paragraph about quantum systems is followed by a 'New reservation' button and view options for 'Card' and 'Table'. A search bar and filters are present above a grid of system cards. Each card shows the system name, status (Online, Offline, or Online - Queue paused), processor type, and quantum volume (Qubits and Quantum volume).

System Name	Status	Processor Type	Qubits	Quantum Volume
ibmq_montreal	Online	Falcon r4	27	128
ibmq_kolkata	Offline	Falcon r5.11	27	128
ibmq_mumbai	Offline	Falcon r5.1	27	128
ibmq_dublin	Online	Falcon r4	27	64
ibm_hanoi	Online - Queue paused	Falcon r5.11	27	64
ibm_cairo	Online	Falcon r5.11	27	64
ibmq_manhattan	Offline	Hummingbird r2	65	32
ibmq_brooklyn	Online - Queue paused	Hummingbird r2	65	32
ibmq_toronto	Online	Falcon r4	27	32
ibmq_sydney	Online	Falcon r4	27	32
ibmq_guadalupe	Online	Falcon r4P	16	32
ibmq_casablanca	Online	Falcon r4H	7	32

Credit: [IBM - Quantum Services](#)

IBM Quantum



Credit: [IBM - Quantum Composer](#)

IBM Quantum

The screenshot displays the IBM Quantum Lab interface. On the left, a file explorer shows a directory structure with files like '01_algorithms_introduction...', '02_vqe_convergence.ipynb', and '07_grover_examples.ipynb'. The main area shows a Jupyter Notebook with the following content:

Boolean Logical Expressions

Qiskit's `Grover` can also be used to perform Quantum Search on an `Oracle` constructed from other means, in addition to DIMACS. For example, the `PhaseOracle` can actually be configured using arbitrary Boolean logical expressions, as demonstrated below.

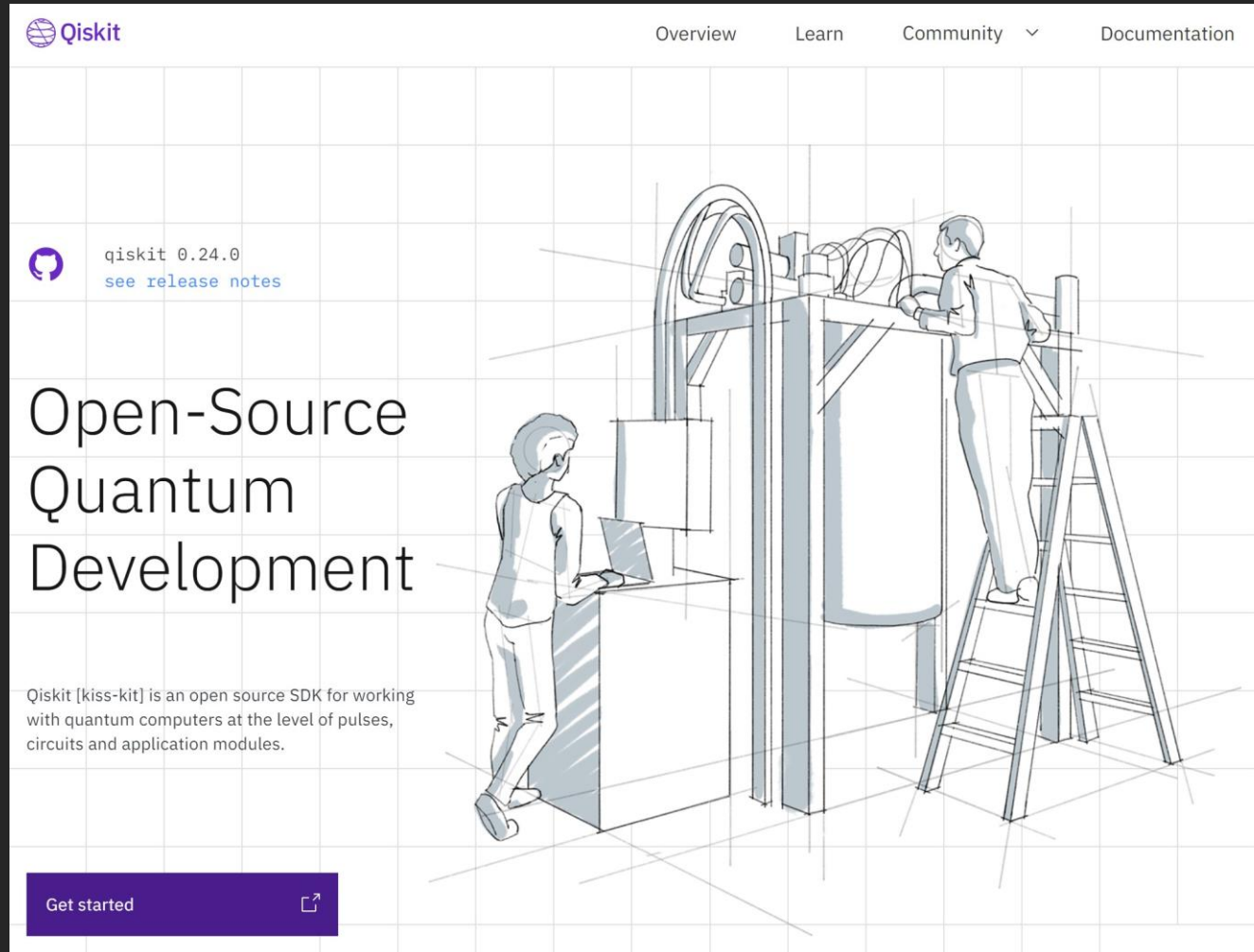
```
[7]: expression = '(w ^ x) & ~(y ^ z) & (x & y & z)'  
try:  
    oracle = PhaseOracle(expression)  
    problem = AmplificationProblem(oracle, is_good_state=oracle.evaluate_bitstring)  
    grover = Grover(quantum_instance=QuantumInstance(Aer.get_backend('aer_simulator'), shots=1024))  
    result = grover.amplify(problem)  
    display(plot_histogram(result.circuit_results[0]))  
except MissingOptionalLibraryError as ex:  
    print(ex)
```

The bar chart below the code shows the probability distribution of results. The x-axis represents bitstrings from 0000 to 1111, and the y-axis represents probabilities from 0.00 to 0.45. The state 1110 has the highest probability, 0.490.

Bitstring	Probability
0000	0.045
0001	0.020
0010	0.020
0011	0.030
0100	0.030
0101	0.030
0110	0.030
0111	0.020
1000	0.030
1001	0.030
1010	0.030
1011	0.020
1100	0.030
1101	0.030
1110	0.490
1111	0.039

Credit: [IBM - Quantum Lab](https://www.ibm.com/quantum/lab)

IBM Quantum



The image shows a screenshot of the Qiskit website homepage. At the top left is the Qiskit logo. To the right are navigation links: Overview, Learn, Community (with a dropdown arrow), and Documentation. Below the navigation is a grid of icons. On the left side of the main content area, there is a section for 'qiskit 0.24.0' with a link to 'see release notes'. The main heading is 'Open-Source Quantum Development'. Below this is a paragraph describing Qiskit as an open source SDK for working with quantum computers at the level of pulses, circuits, and application modules. At the bottom left is a purple 'Get started' button with a share icon. On the right side of the page is a large illustration of two people working on a quantum computer system. One person is standing at a desk with a laptop, and the other is on a ladder working on a large piece of equipment.

Qiskit

Overview Learn Community Documentation

qiskit 0.24.0
[see release notes](#)

Open-Source Quantum Development

Qiskit [kiss-kit] is an open source SDK for working with quantum computers at the level of pulses, circuits and application modules.

Get started

Credit: [IBM - Qiskit](#)

IBM Quantum

The screenshot displays the Qiskit documentation website. At the top, the Qiskit logo is on the left, and navigation links for 'Getting started', 'Tutorials', 'Partners', 'Applications', 'Experiments', 'Resources', and 'Github' are on the right. Below the navigation is a language selector set to 'English'. A search bar labeled 'Search Docs' is present. The main content area features a quantum circuit diagram with six qubits (q0 to q5) and several gates. Below the diagram is the heading 'Qiskit 0.31.0 documentation' followed by a paragraph describing the software. To the right of the main text is a sidebar with three categories: 'Frontmatter' (including 'Quantum computing in a nutshell', 'Introduction to Qiskit', 'Release Notes', 'Contributing to Qiskit', 'Local Configuration', and 'Frequently Asked Questions'), 'Libraries' (including 'Circuit Library'), and 'API References' (including 'Qiskit Terra', 'Qiskit Aer', 'Qiskit Ignis (deprecated)', 'Qiskit Aqua (deprecated)', and 'Qiskit IBM Quantum Provider').

Qiskit

Getting started Tutorials Partners Applications Experiments Resources Github

English

Docs > Qiskit 0.31.0 documentation

Search Docs

Documentation Homepage

Frontmatter

- Quantum computing in a nutshell
- Introduction to Qiskit
- Release Notes
- Contributing to Qiskit
- Local Configuration
- Frequently Asked Questions

Libraries

- Circuit Library

API References

- Qiskit Terra
- Qiskit Aer
- Qiskit Ignis (deprecated)
- Qiskit Aqua (deprecated)
- Qiskit IBM Quantum Provider

Qiskit 0.31.0 documentation

Interested in applications of quantum computing? [Learn more](#)

Interested in running experiments on real quantum hardware? [Learn more](#)

Interested in quantum hardware design? [Learn more](#)

Qiskit 0.31.0 documentation

Qiskit is open-source software for working with quantum computers at the level of circuits, pulses, and algorithms. Additionally, several domain specific application API's exist on top of this core module.

The central goal of Qiskit is to build a software stack that makes it easy for anyone to use quantum computers, regardless of their skill level or area of interest; Qiskit allows one to easily design experiments and applications and run them on real quantum computers and/or classical simulators. Qiskit is already in use around the world by beginners, hobbyists, educators, researchers, and commercial companies.

What is quantum computing?

A quick introduction to quantum computing.

[Get cracking >](#)

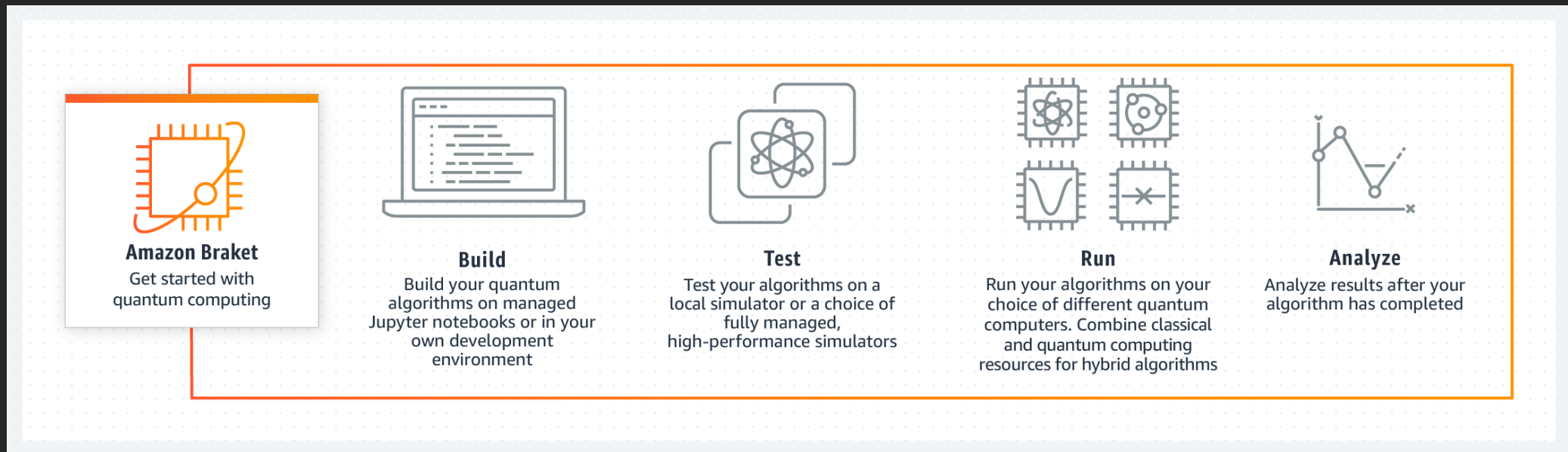
Access to quantum systems

Find out which Qiskit Partners support execution on real quantum services.

[Qiskit Partners >](#)

Credit: [IBM – Qiskit documentation](#)

AWS Braket



Credit: [Amazon AWS - Braket](#)

AWS Braket

Amazon Braket Hardware Providers

Amazon Braket provides AWS customers access to multiple types of quantum computing technologies from quantum hardware providers, including gate-based quantum computers and quantum annealing systems. Learn more about these quantum hardware providers below.



D-Wave's technology uses quantum annealing to solve problems represented as mathematical functions (resembling a landscape of peaks and valleys). Their QPUs are built from a network of interconnected superconducting flux qubits. Each qubit is made from a tiny loop of metal interrupted by a Josephson Junction.

[Learn more »](#)



IONQ's trapped-ion approach to quantum computing starts with ionized ytterbium atoms. Two internal states of these identical atoms make up the qubits, the basic unit of quantum information. The execution of computational tasks is accomplished by programming the sequence of laser pulses used to implement each quantum gate operation.

[Learn more »](#)



Rigetti quantum processors are universal, gate-based machines based on superconducting qubits. The Rigetti Aspen series of chips feature tileable lattices of alternating fixed-frequency and tunable superconducting qubits within a scalable architecture.

[Learn more »](#)

Credit: [Amazon AWS - Braket](#)

PennyLane

The screenshot shows the PennyLane website homepage. At the top, there is a navigation bar with the PennyLane logo, links for "Quantum machine learning", "Install", "Plugins", "Documentation", "QHACK", "FAQ", "Support", and "GitHub". Below the navigation bar, the word "PENNYLANE" is displayed in large, light blue, spaced-out letters. Underneath, a tagline reads: "A cross-platform Python library for differentiable programming of quantum computers. Train a quantum computer the same way as a neural network." Three main content cards are featured: "Learn" (with a video thumbnail), "Play" (with a code snippet), and "Hack" (with a QHACK 2021 logo). Below these cards, a section titled "PennyLane supports a growing ecosystem, including a wide range of quantum hardware and machine learning libraries" displays logos for XANADU, AWS, IBM, Google, rigetti, Microsoft, ZAPATA, QAO, TensorFlow, and PyTorch.

PENNYLANE

Quantum machine learning

Install Plugins Documentation QHACK

? FAQ Support GitHub

A cross-platform Python library for differentiable programming of quantum computers. Train a quantum computer the same way as a neural network.

Learn
Sit back and learn about the field of quantum machine learning, explore key concepts, and view our selection of curated videos.
Quantum machine learning >>

Play
Tutorials to introduce core QML concepts, including quantum nodes, optimization, and devices, via easy-to-follow examples.
Demos >>

Hack
Join us for QHACK, the quantum machine learning hackathon. Feb 17-26th 2021.
Sign up >>

PennyLane supports a growing ecosystem, including a wide range of quantum hardware and machine learning libraries

XANADU AWS IBM Google rigetti

Microsoft ZAPATA QAO TensorFlow PyTorch

Credit: [Xanadu - PennyLane](#)

Google Cirq



Cirq

An open source framework for programming quantum computers

Cirq is a Python software library for writing, manipulating, and optimizing quantum circuits, and then running them on quantum computers and quantum simulators. Cirq provides useful abstractions for dealing with today's noisy intermediate-scale quantum computers, where details of the hardware are vital to achieving state-of-the-art results.

[Get started with Cirq](#)

[GitHub repository](#)

```
import cirq

# Pick a qubit.
qubit = cirq.GridQubit(0, 0)

# Create a circuit
circuit = cirq.Circuit(
    cirq.X(qubit)**0.5, # Square root of NOT.
    cirq.measure(qubit, key='m') # Measurement.
)
print("Circuit:")
print(circuit)

# Simulate the circuit several times.
simulator = cirq.Simulator()
result = simulator.run(circuit, repetitions=20)
print("Results:")
print(result)
```

Credit: [Google Quantum AI - Cirq](#)

TensorFlow Quantum

TensorFlow Quantum is a library for hybrid quantum-classical machine learning.

TensorFlow Quantum (TFQ) is a [quantum machine learning](#) library for rapid prototyping of hybrid quantum-classical ML models. Research in quantum algorithms and applications can leverage Google's quantum computing frameworks, all from within TensorFlow.

TensorFlow Quantum focuses on *quantum data* and building *hybrid quantum-classical models*. It integrates quantum computing algorithms and logic designed in [Cirq](#), and provides quantum computing primitives compatible with existing TensorFlow APIs, along with high-performance quantum circuit simulators. Read more in the [TensorFlow Quantum white paper](#).

Start with the [overview](#), then run the [notebook tutorials](#).

```
# A hybrid quantum-classical model.
model = tf.keras.Sequential([
    # Quantum circuit data comes in inside of tensors.
    tf.keras.Input(shape=(), dtype=tf.dtypes.string),

    # Parametrized Quantum Circuit (PQC) provides output
    # data from the input circuits run on a quantum computer.
    tfq.layers.PQC(my_circuit, [cirq.Z(q1), cirq.X(q0)]),

    # Output data from quantum computer passed through model.
    tf.keras.layers.Dense(50)
])
```

Credit: [Google – TensorFlow Quantum](#)

D-Wave Ocean/Leap

The screenshot displays the Leap IDE interface. On the left, a Python script named `maximum_cut.py` is shown. The script creates a graph with 5 nodes and 6 edges, sets up a QUBO dictionary, and runs it on a D-Wave QPU. The output shows a table of solutions and their energy values.

```
29 # Create empty graph
30 G = nx.Graph()
31
32 # Add edges to the graph (also adds nodes)
33 G.add_edges_from([(1,2),(1,3),(2,4),(3,4),(3,5),(4,5)])
34
35 # ----- Set up our QUBO dictionary -----
36
37 # Initialize our Q matrix
38 Q = defaultdict(int)
39
40 # Update Q matrix for every edge in the graph
41 for i, j in G.edges:
42     Q[(i,i)]+= -1
43     Q[(j,j)]+= -1
44     Q[(i,j)]+= 2
45
46 # ----- Run our QUBO on the QPU -----
47 # Set up QPU parameters
48 chainstrength = 8
49 numruns = 10
50
51 # Run the QUBO on the solver from your config file
52 sampler = EmbeddingComposite(DWaveSampler({"qpu": True}))
53 response = sampler.sample_qubo(Q,
54                               chain_strength=chainstrength,
55                               num_reads=numruns,
56                               label='Example - Maximum Cut')
57 dwave.inspector.show(response)
```

The output table shows the following results:

Set 0	Set 1	Energy	Cut Size
[1, 4, 5]	[2, 3]	-5.0	5
[2, 3, 5]	[1, 4]	-5.0	5
[1, 4]	[2, 3, 5]	-5.0	5
[2, 3]	[1, 4, 5]	-5.0	5
[4, 5]	[1, 2, 3]	-3.0	3
[3]	[1, 2, 4, 5]	-3.0	3
[1, 3, 5]	[2, 4]	-3.0	3
[1, 3]	[2, 4, 5]	-3.0	3

The Problem Inspector shows the source graph (Force Directed) and the target QPU graph. The console shows the command executed: `Leap IDE /workspace/maximum-cut $ /usr/local/bin/python /workspace/maximum-cut/maximum_cut.py`. The status bar at the bottom indicates the current file is `dwave-examples/maximum-cut` and the Python version is 3.7.10 64-bit.

Credit: [D-Wave - Leap](#)