takatoa

```
OPENQASM 2.0;
include "qelib1.inc";

qreg q[5];
creg c[3];

h q[0];
h q[1];
h q[2];
h q[1];
cx q[2],q[3];
cu1(pi/2) q[1],q[0];
cx q[2],q[4];
h q[0];
cu1(pi/4) q[1],q[2];
cu1(pi/2) q[0],q[2];
h q[2];
measure q[0] -> c[0];
measure q[1] -> c[1];
measure q[2] -> c[2];
```

Quantum Artificial Intelligence meets GIS

THE FUTURE IS HERE

Roland Degelmann
– head of takatoa –
Freising, Germany

takatoa

QAI meets GIS – in a nutshell

source: ibm.com

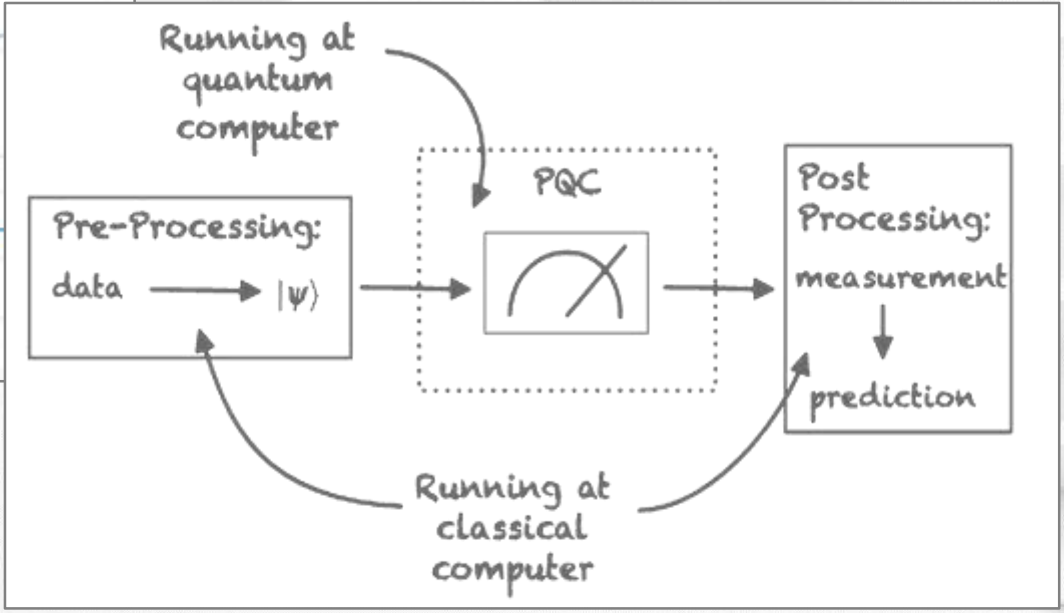
Geo-Information-System (GIS)

Extract Data from various Layers

Train the QAI System

Use the trained System

Submit Results to Feature/Image Layer



source: pyqml.com

Quantum timetable

1900
Planck proposes that radiation comes in discrete amounts, or is quantized

1924
Louis de Broglie proposes that matter has wave properties

1960
First laser built

1981
Richard Feynman invents the concept of a quantum computer

1994
Peter Shor invents the first "useful" quantum algorithm to crack prime number-based cryptographies

1905
Einstein suggests a quantum of light (the photon)

1926
Erwin Schroedinger develops wave mechanics

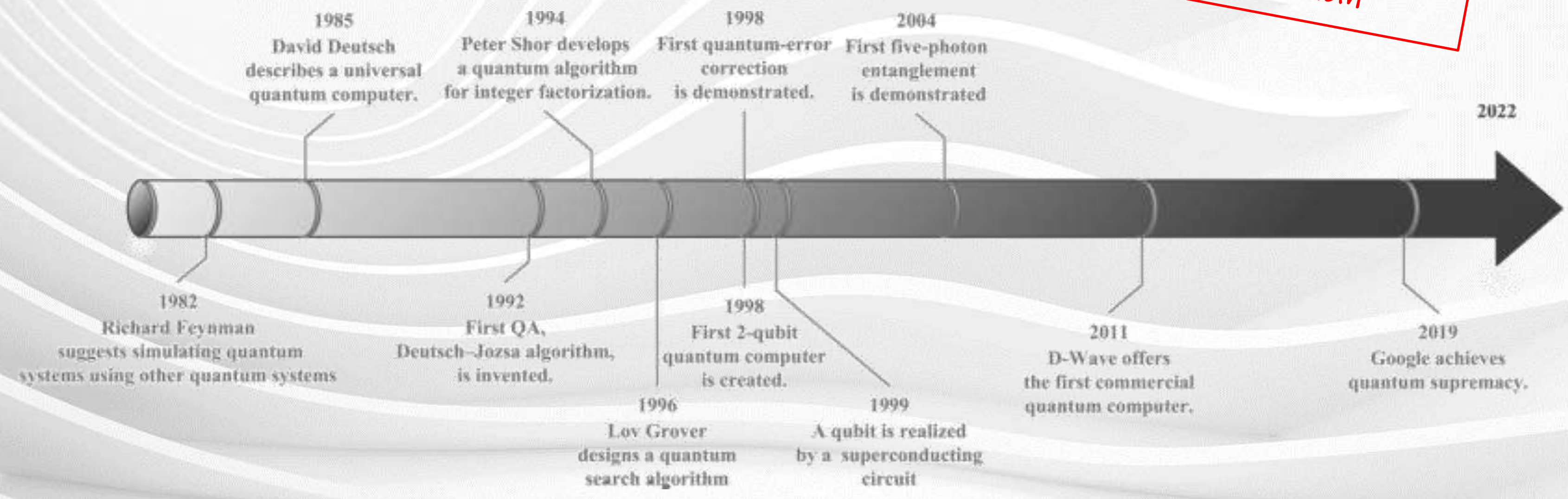
1973
GPS launched by US Department of Defense

1981
Alain Aspect demonstrates the existence of the entanglement phenomenon

2001
First execution of Shor algorithm on a quantum device, Stanford University

source: atos.net

> If you think you understand quantum mechanics, you don't understand quantum mechanics.
> Richard P. Feynman



1982
Richard Feynman suggests simulating quantum systems using other quantum systems

1985
David Deutsch describes a universal quantum computer.

1992
First QA, Deutsch–Jozsa algorithm, is invented.

1994
Peter Shor develops a quantum algorithm for integer factorization.

1996
Lov Grover designs a quantum search algorithm

1998
First quantum-error correction is demonstrated.

1998
First 2-qubit quantum computer is created.

1999
A qubit is realized by a superconducting circuit

2004
First five-photon entanglement is demonstrated

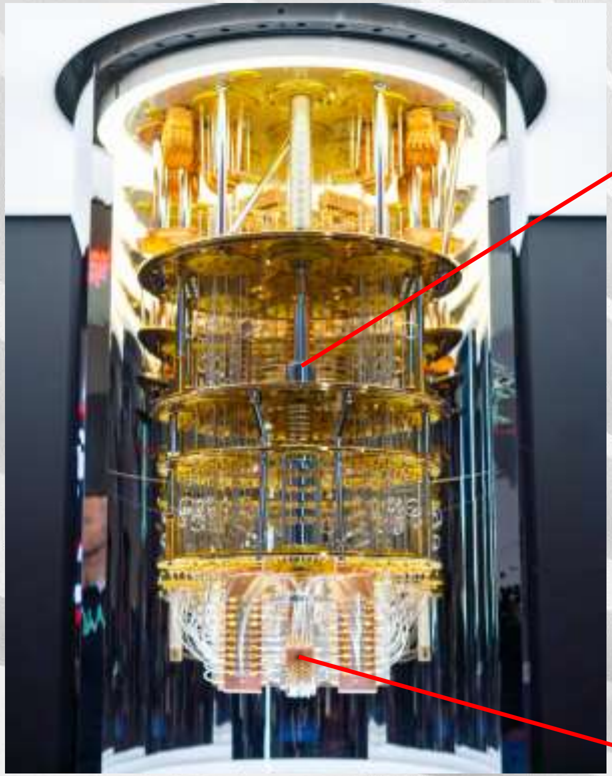
2011
D-Wave offers the first commercial quantum computer.

2019
Google achieves quantum supremacy.

source: sciencedirect.com

The „Quantum Reactor" – NISQ technology

Quantum computer

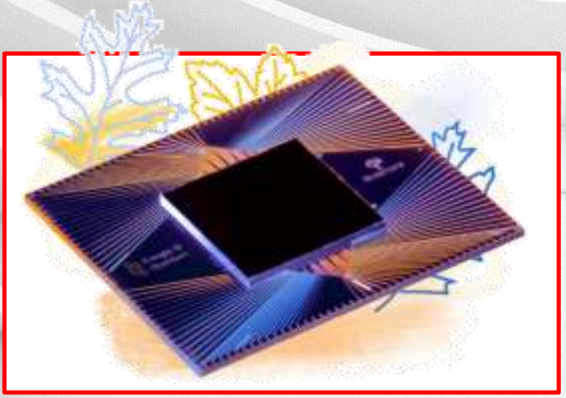


Erik Lucero
Leading Engineer at Google
Google IO 2021 | Quantum AI Campus

"this is only the fridge"

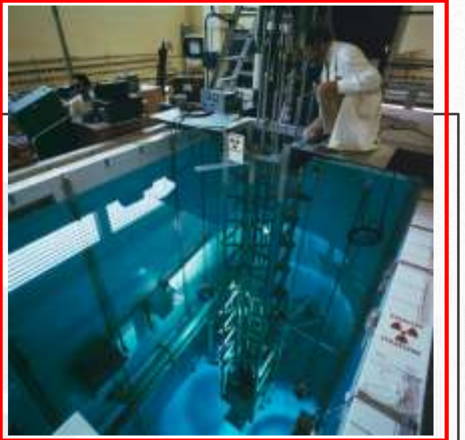
target: close to 0°K (~ -273.15C)

Quantum processor
(Google Sycamore 53 qubits)



source: ibm.com / google.ai

Nuclear power plant



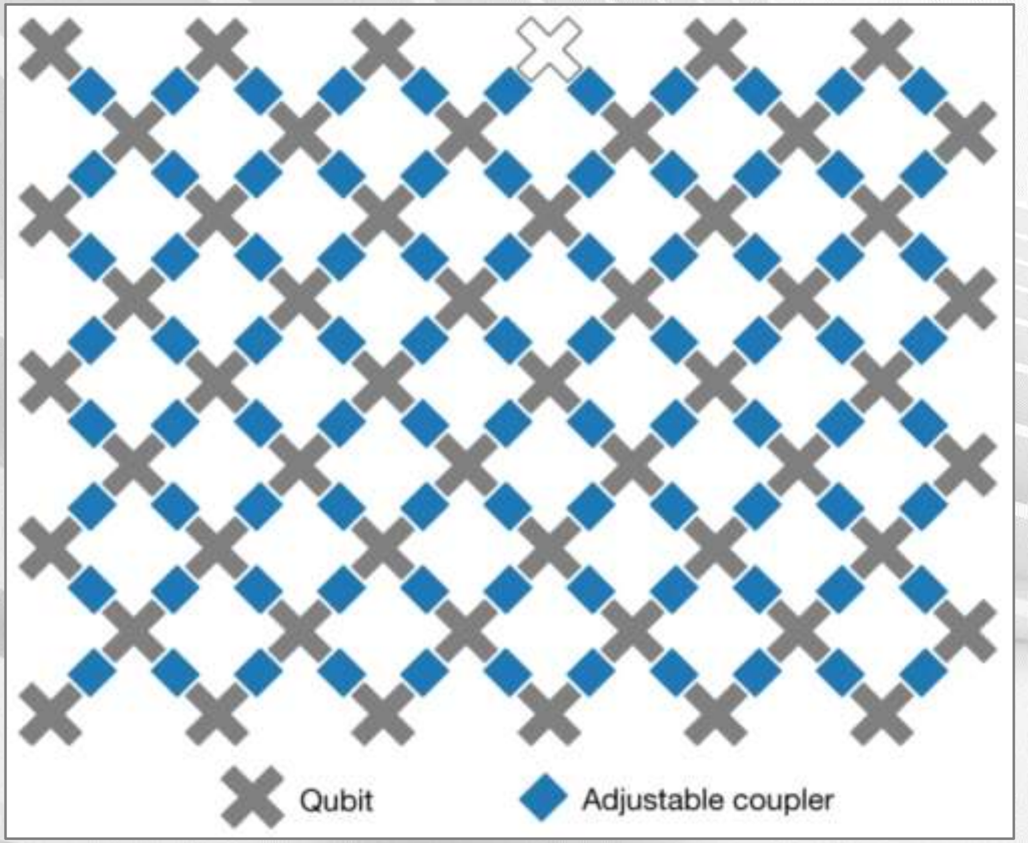
Nuclear reactor

Cooling towers

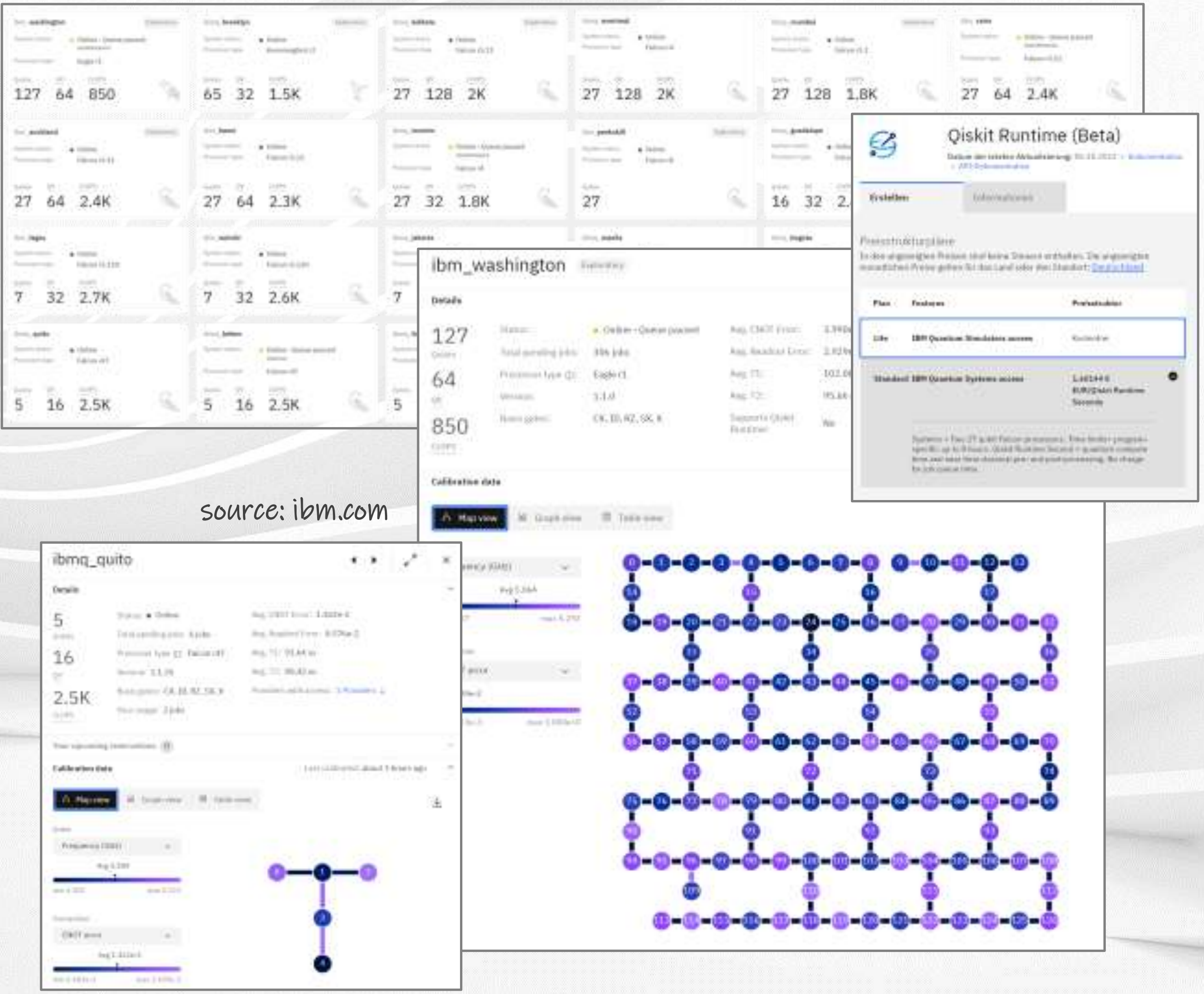
source: www.rentokil-ths.de / de.wikipedia.org

Quantum processors – Google and IBM (and many others)

NISQ – Noisy Intermediate-Scale Quantum

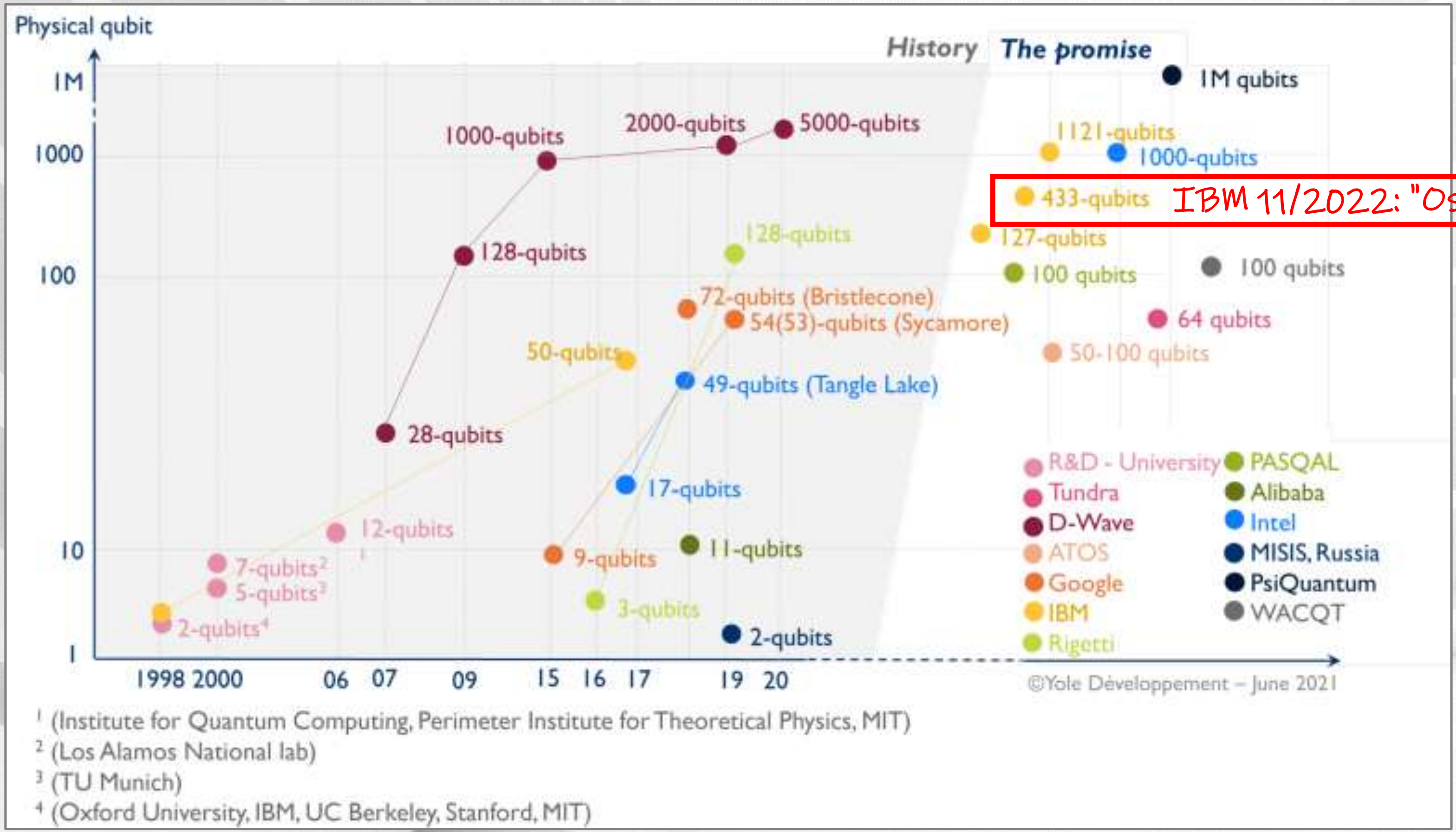


source: sycamore - google.com



source: ibm.com

Physical Qubit Roadmap for Quantum Computer

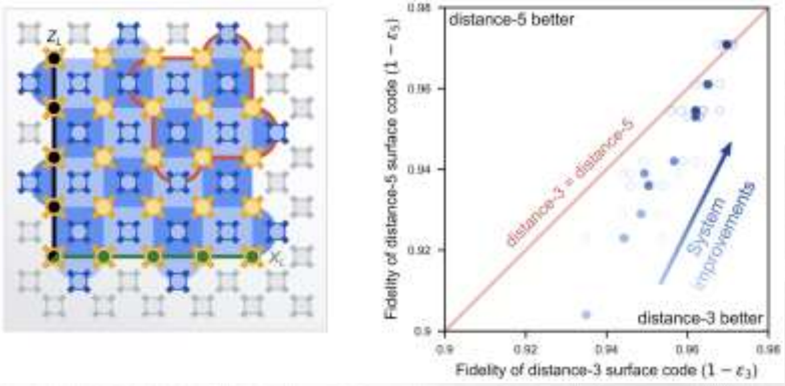


[1] (Institute for Quantum Computing, Perimeter Institute for Theoretical Physics, MIT)
[2] (Los Alamos National lab)
[3] (TU Munich)
[4] (Oxford University, IBM, UC Berkeley, Stanford, MIT)

IBM 11/2022: "Osprey"

source: Quantum Technologies2021 yole.fr 2021

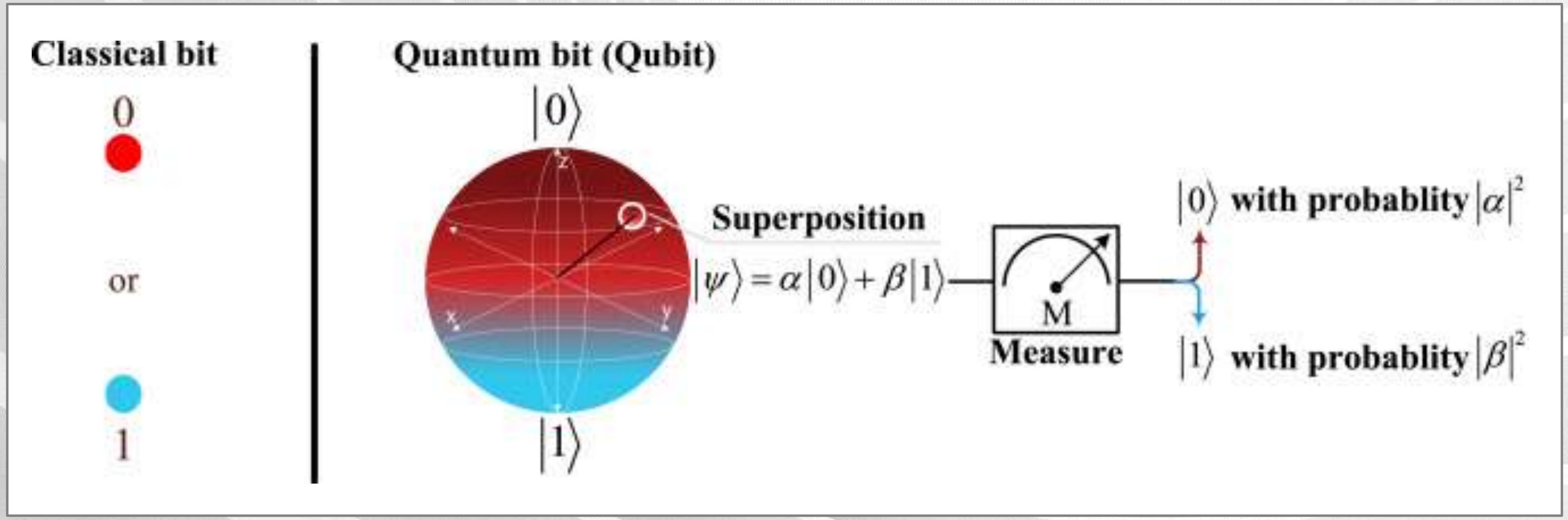
Google 02/2023:
Quantum error correction using logical qubits

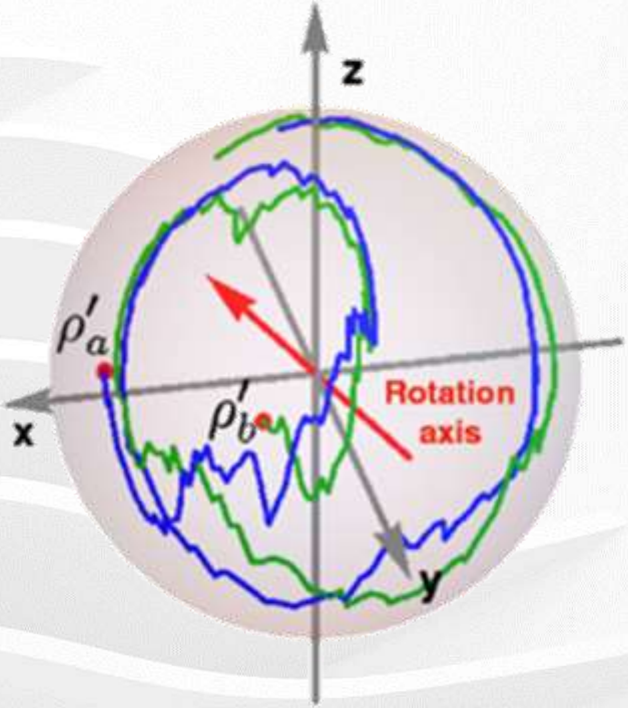


source:
ai.googleblog.com/2023/02/suppressing-quantum-errors-by-scaling.html

Bit – Qubit

source: sciencedirect.com





With n bits one of 2^n states can be represented at the same time.
With n qubits 2^n states can be represented at the same time.

The information contained in a single qubit can be described by a linear combination of |0⟩ and |1⟩:

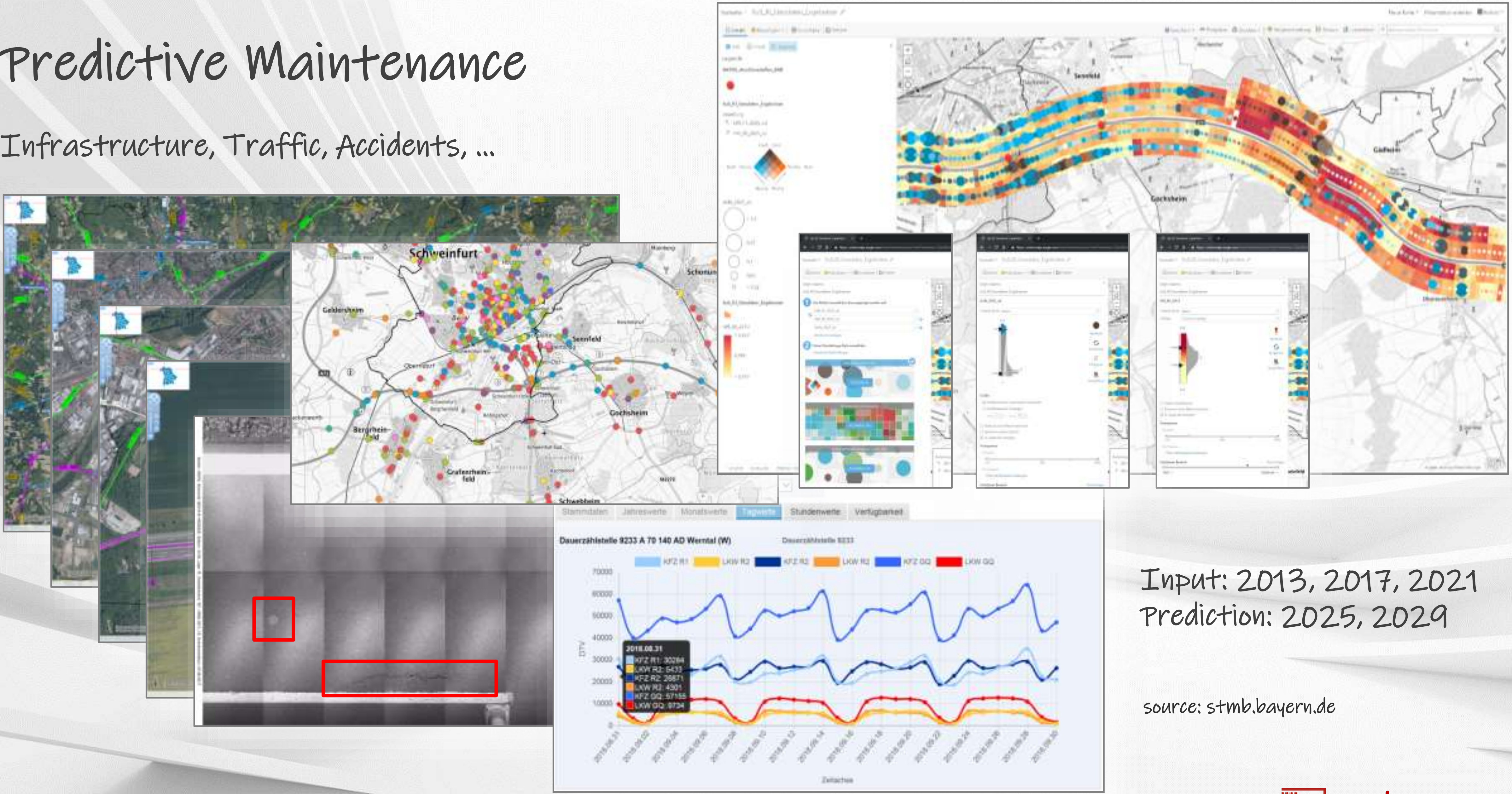
$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ with $|\alpha|^2 + |\beta|^2 = 1.$

> What we need to bear in mind is,
> that quantum technology is not evolution, it is revolution.
>
> Dr. Hartmut Neven; Google Quantum AI

Abacus ≃ Classical computer ≠ Quantum computer

Predictive Maintenance

Infrastructure, Traffic, Accidents, …

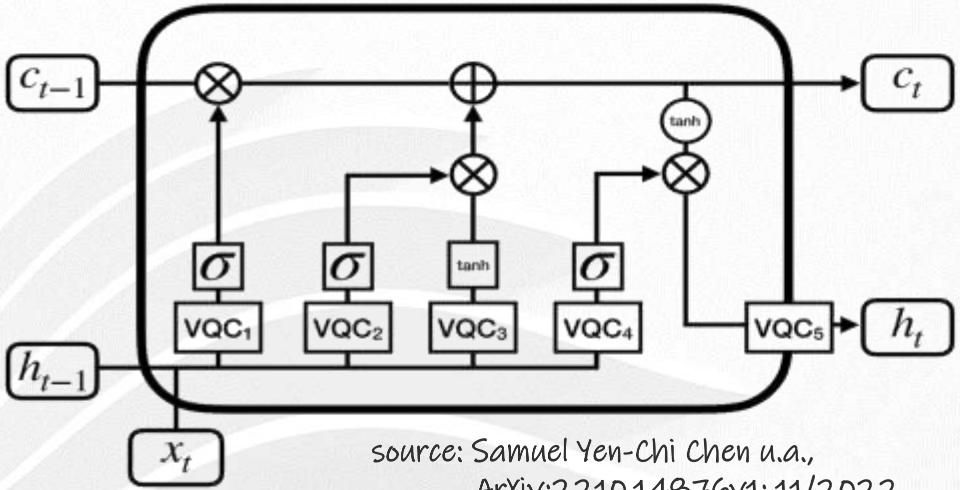


Input: 2013, 2017, 2021
Prediction: 2025, 2029

source: stmb.bayern.de

Predictive Maintenance using LSTM

Long short-term memory (LSTM) enables neural networks to have a kind of memory of previous experiences. It realizes a short-term memory that lasts for a long time because the principal behavior of the network is encoded in the weights.



source: Samuel Yen-Chi Chen u.a., ArXiv:2210.14876v1; 11/2022

Value synthesis of condition recording and assessment
in Germany (ZEB = Zustandserfassung und -bewertung)

Griffigkeit

Längsebenheit

Querebenheit

Substanzmerkmale (Oberfläche)

Gebrauchswert

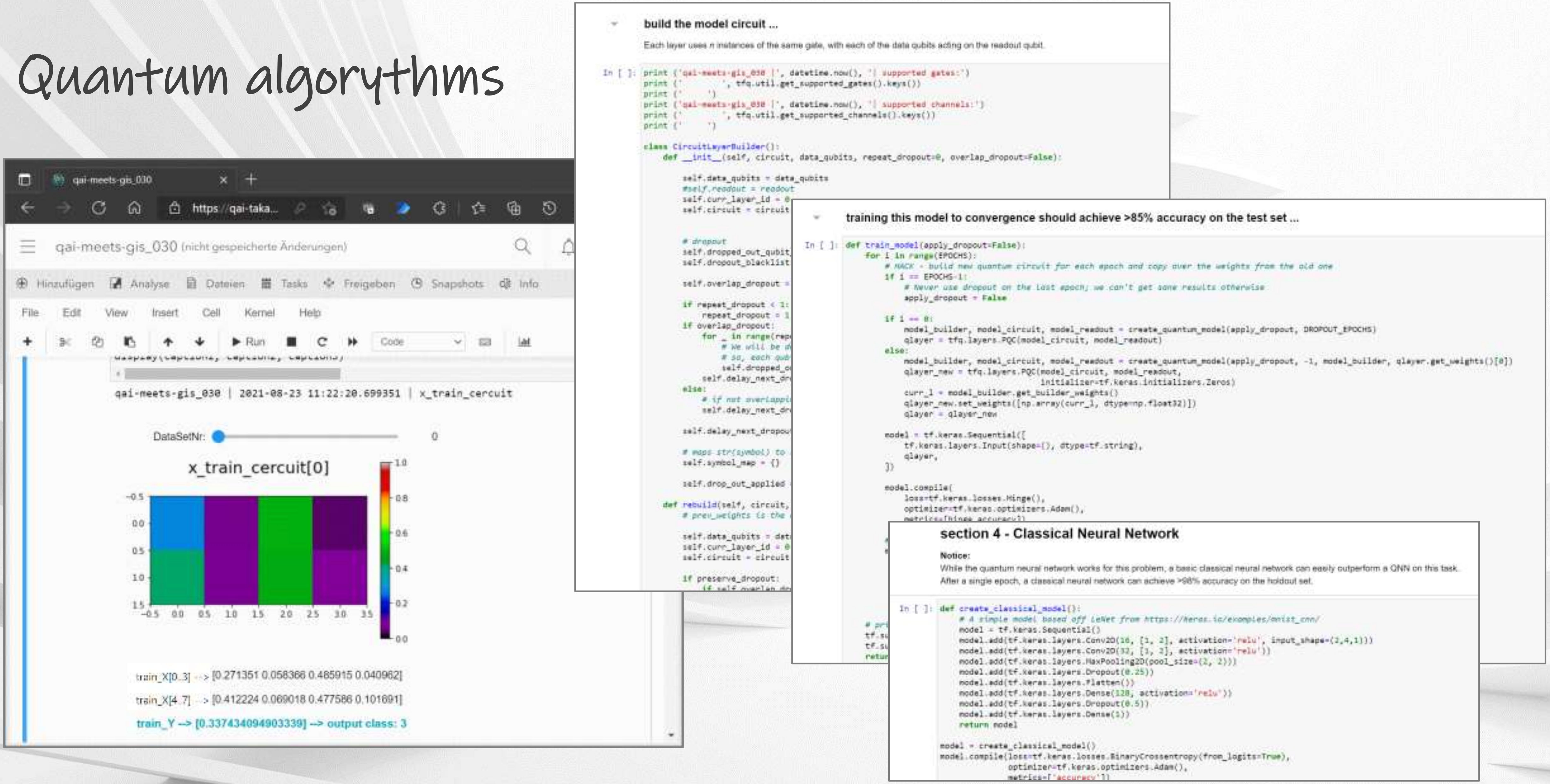
Substanzwert

Gesamtwert

source: www.bmvi.de

data source:
stmb.bayern.de

Quantum algorythms



```
qai-meets-gis_030 | 2021-08-23 11:22:20.699351 | x_train_cercuit
```

train_X[0..3] --> [0.271351 0.058366 0.485915 0.040962]

train_X[4..7] --> [0.412224 0.069018 0.477586 0.101691]

train_Y --> [0.337434094903339] --> output class: 3

build the model circuit ...

Each layer uses *n* instances of the same gate, with each of the data qubits acting on the readout qubit.

```
In [ ]: print ('qai-meets-gis_030 |', datetime.now(), '| supported gates:')
        print ('        ', tfq.util.get_supported_gates().keys())
        print ('     ')
        print ('qai-meets-gis_030 |', datetime.now(), '| supported channels:')
        print ('        ', tfq.util.get_supported_channels().keys())
        print ('     ')

        class CircuitLayerBuilder():
            def __init__(self, circuit, data_qubits, repeat_dropout=0, overlap_dropout=False):

                self.data_qubits = data_qubits
                #self.readout = readout
                self.curr_layer_id = 0
                self.circuit = circuit


                # dropout
                self.dropped_out_qubit
                self.dropout_blacklist

                self.overlap_dropout =

                if repeat_dropout < 1:
                    repeat_dropout = 1
                if overlap_dropout:
                    for _ in range(repe
                        # We will be d
                        # so, each qub
                        self.dropped_o
                    self.delay_next_dr
                else:
                    # if not overlappi
                    self.delay_next_dr

                self.delay_next_dropou

                # maps str(symbol) to
                self.symbol_map = {}

                self.drop_out_applied

            def rebuild(self, circuit,
                # prev_weights is the

                self.data_qubits = dat
                self.curr_layer_id = 0
                self.circuit = circuit

                if preserve_dropout:
```

training this model to convergence should achieve >85% accuracy on the test set ...

```
In [ ]: def train_model(apply_dropout=False):
            for i in range(EPOCHS):
                # HACK - build new quantum circuit for each epoch and copy over the weights from the old one
                if i == EPOCHS-1:
                    # Never use dropout on the last epoch; we can't get sone results otherwise
                    apply_dropout = False

                if i == 0:
                    model_builder, model_circuit, model_readout = create_quantum_model(apply_dropout, DROPOUT_EPOCHS)
                    qlayer = tfq.layers.PQC(model_circuit, model_readout)
                else:
                    model_builder, model_circuit, model_readout = create_quantum_model(apply_dropout, -1, model_builder, qlayer.get_weights()[0])
                    qlayer_new = tfq.layers.PQC(model_circuit, model_readout,
                                                initializer=tf.keras.initializers.Zeros)
                    curr_l = model_builder.get_builder_weights()
                    qlayer_new.set_weights([np.array(curr_l, dtype=np.float32)])
                    qlayer = qlayer_new

                model = tf.keras.Sequential([
                    tf.keras.layers.Input(shape=(), dtype=tf.string),
                    qlayer,
                ])

                model.compile(
                    loss=tf.keras.losses.Hinge(),
                    optimizer=tf.keras.optimizers.Adam(),
                    metrics=[hinge_accuracy])



            # pri
            tf.su
            tf.su
            retur
```

section 4 - Classical Neural Network

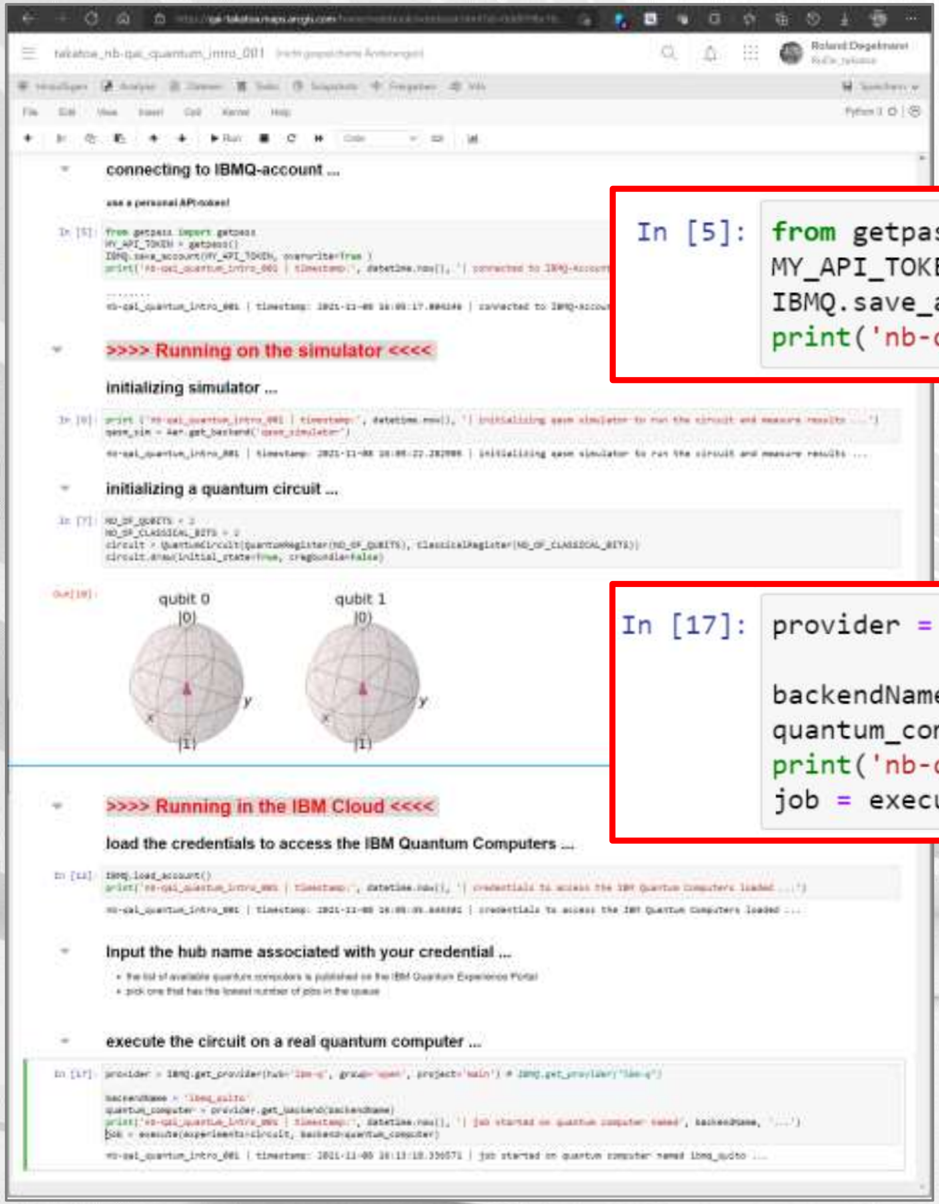
Notice:

While the quantum neural network works for this problem, a basic classical neural network can easily outperform a QNN on this task.
After a single epoch, a classical neural network can achieve >98% accuracy on the holdout set.

```
In [ ]: def create_classical_model():
            # A simple model based off LeNet from https://keras.io/examples/mnist_cnn/
            model = tf.keras.Sequential()
            model.add(tf.keras.layers.Conv2D(16, [1, 2], activation='relu', input_shape=(2,4,1)))
            model.add(tf.keras.layers.Conv2D(32, [1, 2], activation='relu'))
            model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
            model.add(tf.keras.layers.Dropout(0.25))
            model.add(tf.keras.layers.Flatten())
            model.add(tf.keras.layers.Dense(128, activation='relu'))
            model.add(tf.keras.layers.Dropout(0.5))
            model.add(tf.keras.layers.Dense(1))
            return model

        model = create_classical_model()
        model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
                      optimizer=tf.keras.optimizers.Adam(),
                      metrics=['accuracy'])
```

Connect to IBMQ Quantum



connecting to IBMQ-account ...

use a personal API-token!

>>>> Running on the simulator <<<<

initializing simulator ...

initializing a quantum circuit ...

>>>> Running in the IBM Cloud <<<<

load the credentials to access the IBM Quantum Computers ...

Input the hub name associated with your credential ...

- the list of available quantum computers is published on the IBM Quantum Experience Portal
- pick one that has the lowest number of jobs in the queue

execute the circuit on a real quantum computer ...

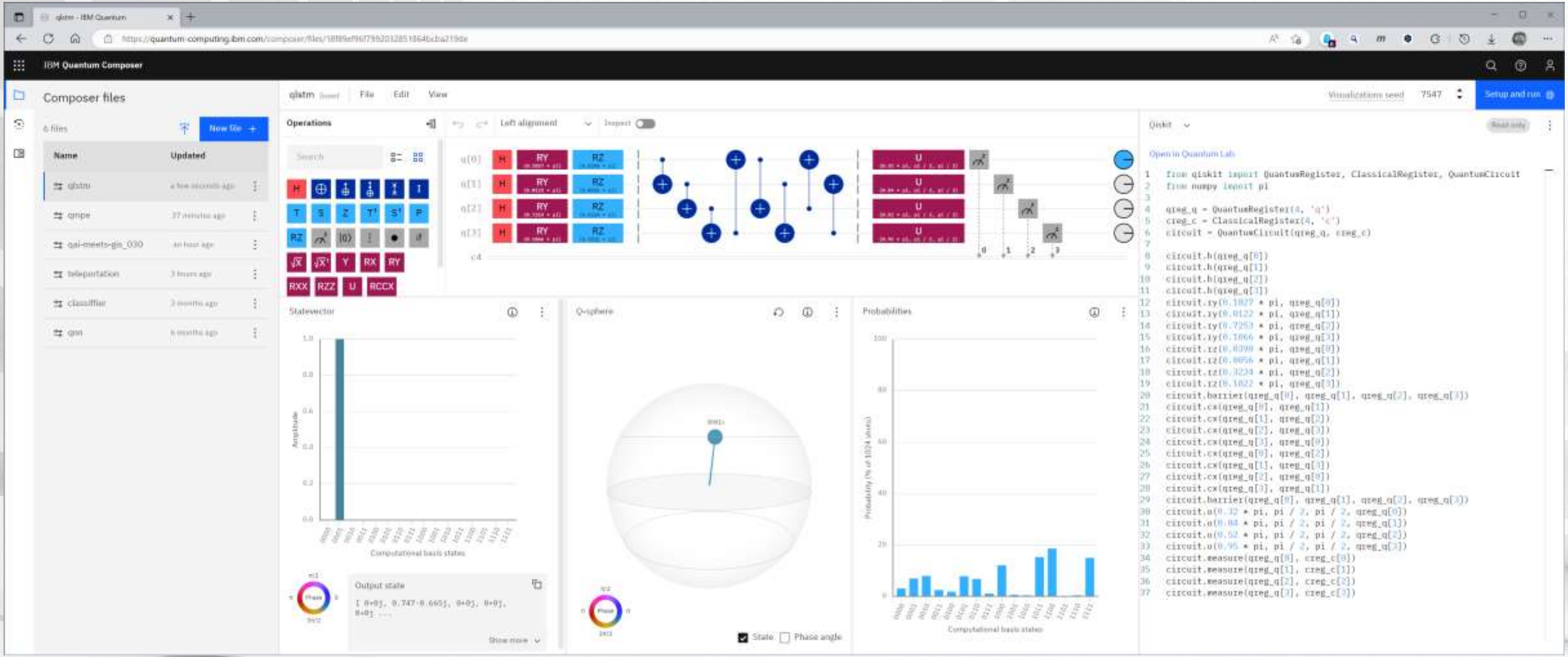
```
In [5]: from getpass import getpass
        MY_API_TOKEN = getpass()
        IBMQ.save_account(MY_API_TOKEN, overwrite=True )
        print('nb-qai_quantum_intro_001 | timestamp:', datetime.now(), '| connected to IBMQ-Account ...')
```

```
In [17]: provider = IBMQ.get_provider(hub='ibm-q', group='open', project='main')

         backendName = 'ibmq_quito'
         quantum_computer = provider.get_backend(backendName)
         print('nb-qai_quantum_intro_001 | timestamp:', datetime.now(), '| job started ...')
         job = execute(experiments=circuit, backend=quantum_computer)
```

Quantum Long short-term memory

IBM Quantum Composer



```
[ 0+0j, 0.747-0.665j, 0+0j, 0+0j,
0+0j ...
```

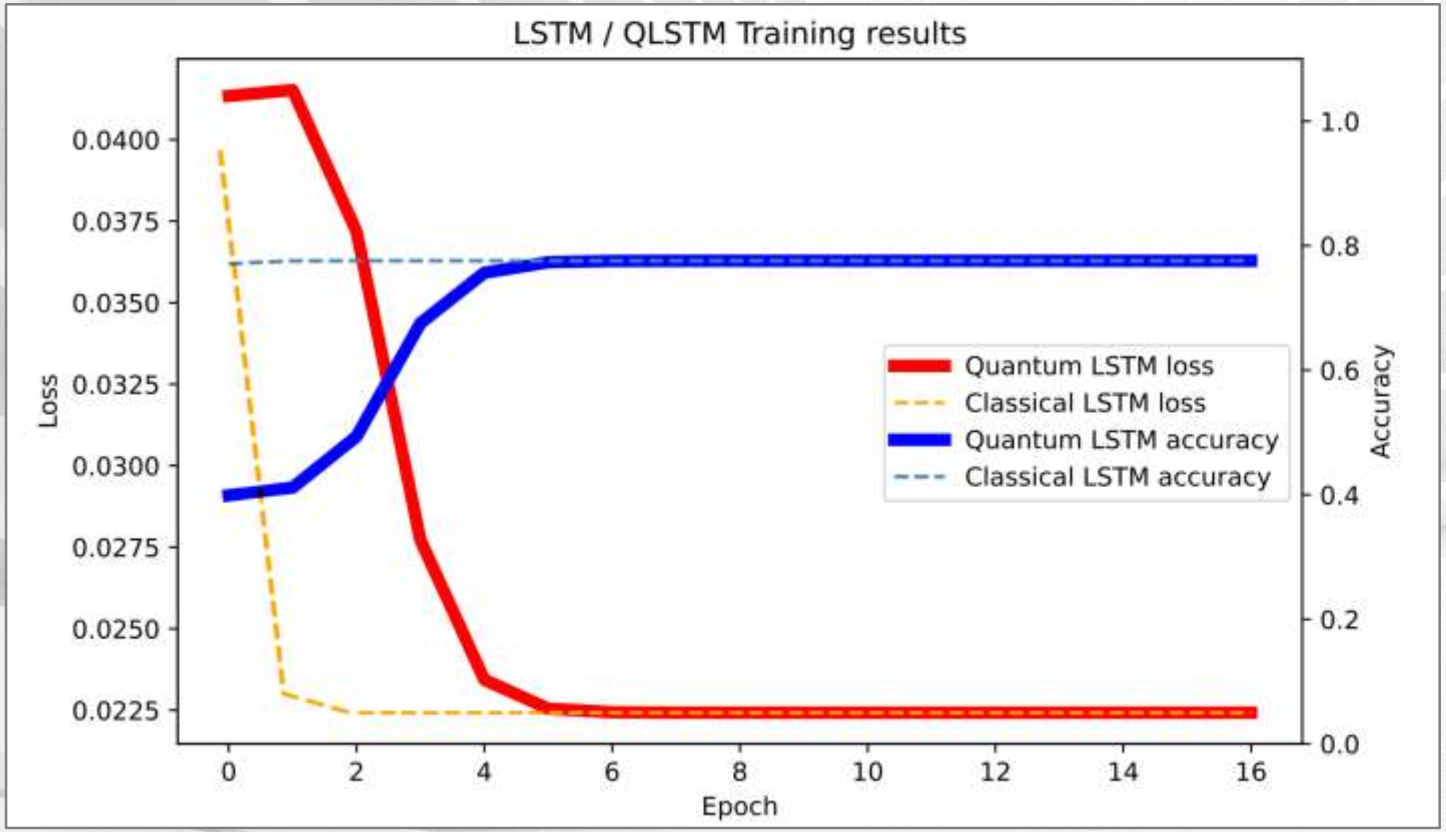
```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(4, 'q')
creg_c = ClassicalRegister(4, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

circuit.h(qreg_q[0])
circuit.h(qreg_q[1])
circuit.h(qreg_q[2])
circuit.h(qreg_q[3])
circuit.ry(0.1827 * pi, qreg_q[0])
circuit.ry(0.0122 * pi, qreg_q[1])
circuit.ry(0.7253 * pi, qreg_q[2])
circuit.ry(0.1866 * pi, qreg_q[3])
circuit.rz(0.0398 * pi, qreg_q[0])
circuit.rz(0.0056 * pi, qreg_q[1])
circuit.rz(0.3234 * pi, qreg_q[2])
circuit.rz(0.1022 * pi, qreg_q[3])
circuit.barrier(qreg_q[0], qreg_q[1], qreg_q[2], qreg_q[3])
circuit.cx(qreg_q[0], qreg_q[1])
circuit.cx(qreg_q[1], qreg_q[2])
circuit.cx(qreg_q[2], qreg_q[3])
circuit.cx(qreg_q[3], qreg_q[0])
circuit.cx(qreg_q[0], qreg_q[2])
circuit.cx(qreg_q[1], qreg_q[3])
circuit.cx(qreg_q[2], qreg_q[0])
circuit.cx(qreg_q[3], qreg_q[1])
circuit.barrier(qreg_q[0], qreg_q[1], qreg_q[2], qreg_q[3])
circuit.u(0.32 * pi, pi / 2, pi / 2, qreg_q[0])
circuit.u(0.84 * pi, pi / 2, pi / 2, qreg_q[1])
circuit.u(0.52 * pi, pi / 2, pi / 2, qreg_q[2])
circuit.u(0.95 * pi, pi / 2, pi / 2, qreg_q[3])
circuit.measure(qreg_q[0], creg_c[0])
circuit.measure(qreg_q[1], creg_c[1])
circuit.measure(qreg_q[2], creg_c[2])
circuit.measure(qreg_q[3], creg_c[3])
```

Predictive Maintenance – Results

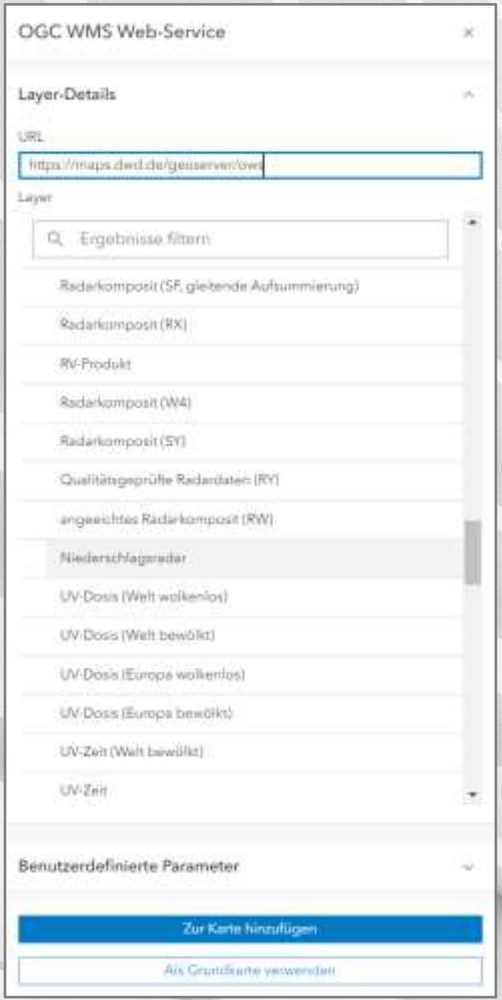
The calculations performed so far basically confirm the usability of the above approaches. They show good learning ability and high result stability.



Example of training results of a comparative computation run (conventional / quantum assisted) which was used to predict temporal developments of state values.
The calculation was performed with 3872 training data sets with eight data each (four input values each for two time points t-1 and t), and with one result value each for the time point t+1 known for the training.
967 additional data sets were used for the examination of the results.

QAI on Rainfall radar images

https://maps.dwd.de/geoserver/ows?service=wms&version=1.3.0&request=GetCapabilities



OGC WMS Web-Service

Layer-Details

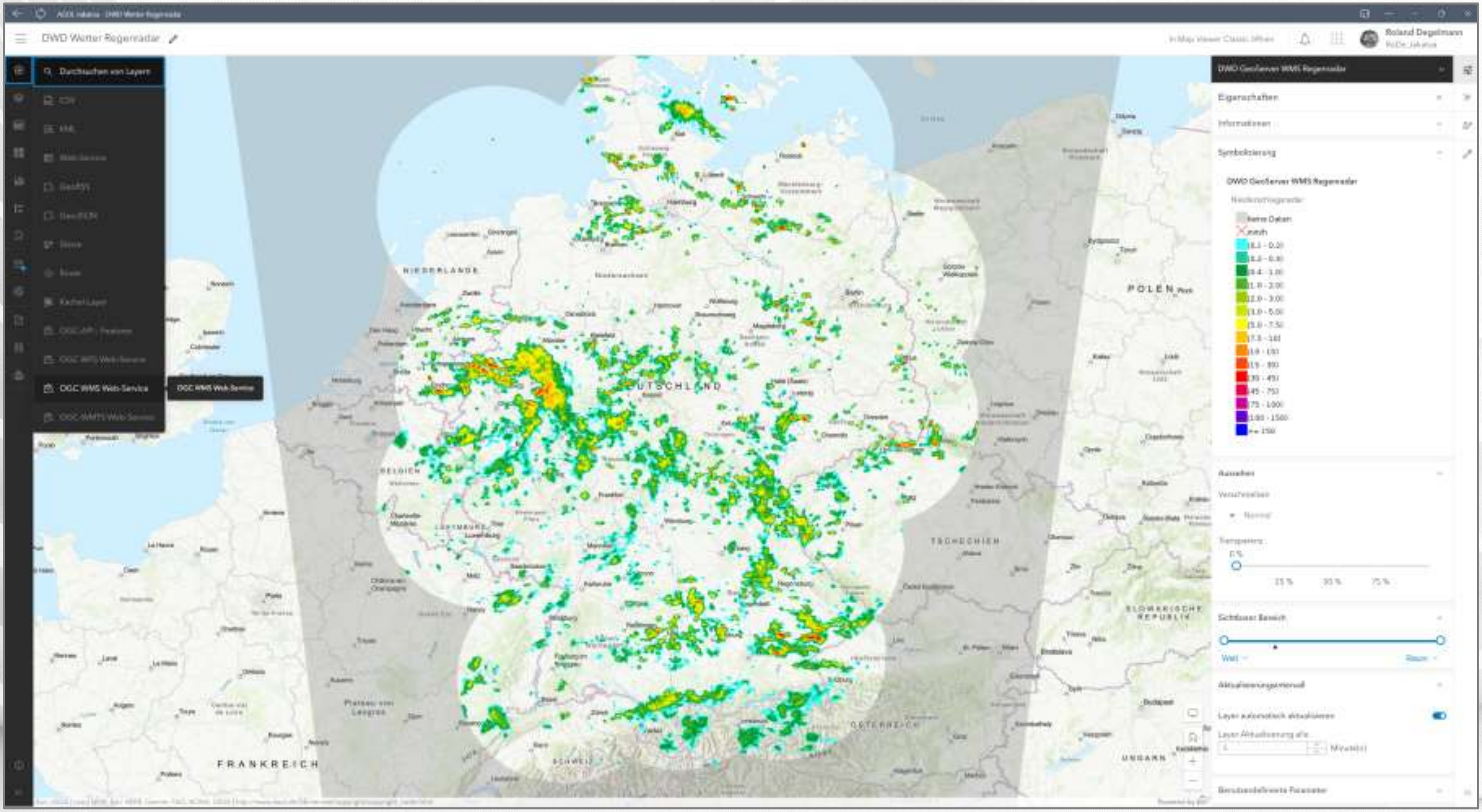
URL: https://maps.dwd.de/geoserver/ows

Layer

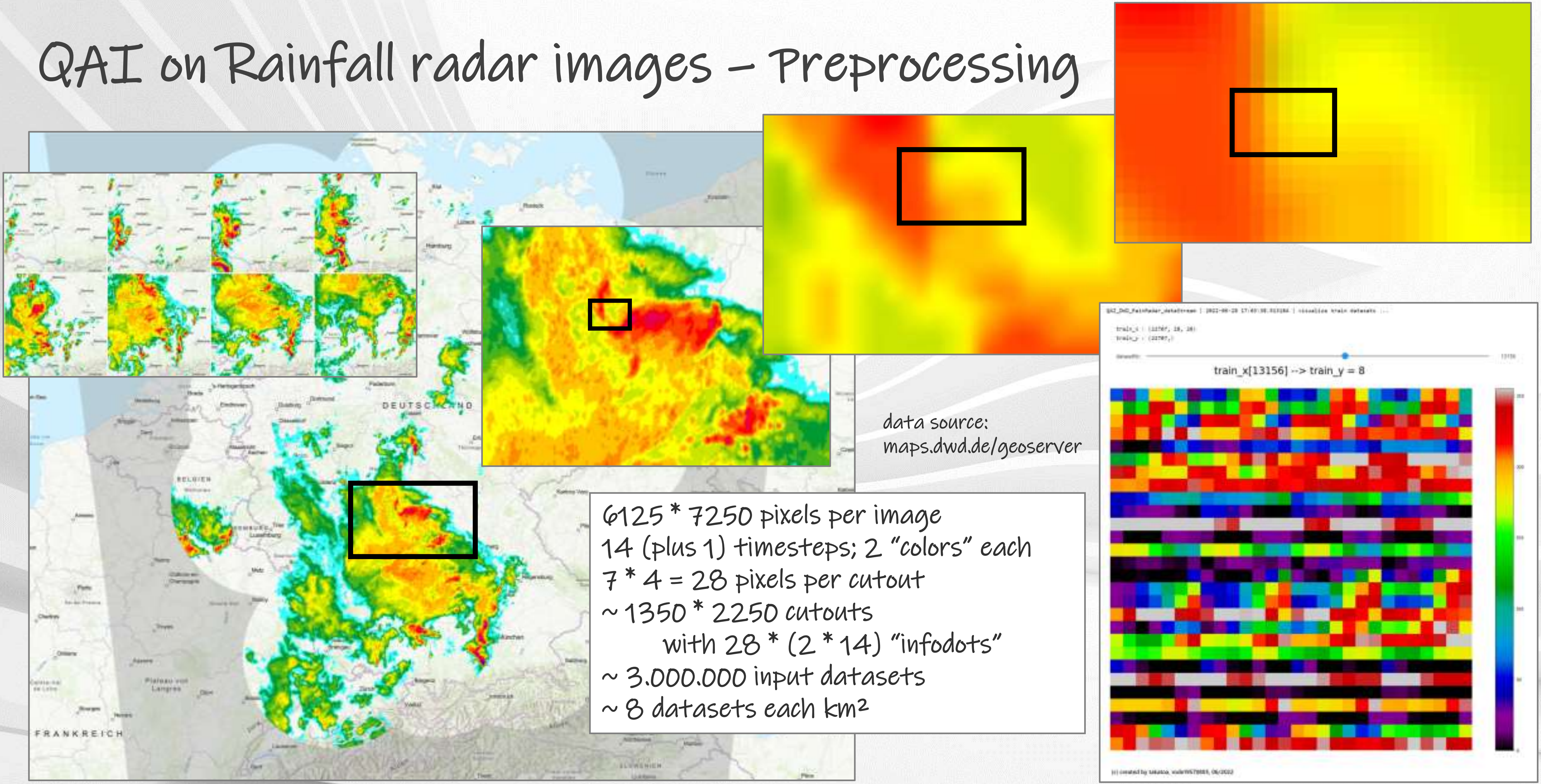
Ergebnisse filtern

- Radarkomposit (SF, gleitende Aufsummierung)
- Radarkomposit (RX)
- RV-Produkt
- Radarkomposit (W4)
- Radarkomposit (SY)
- Qualitätsgeprüfte Radardaten (RY)
- angeeichtes Radarkomposit (RW)
- Niederschlagsradar
- UV-Dosis (Welt wolkenlos)
- UV-Dosis (Welt bewölkt)
- UV-Dosis (Europa wolkenlos)
- UV-Dosis (Europa bewölkt)
- UV-Zeit (Welt bewölkt)
- UV-Zeit

Benutzerdefinierte Parameter



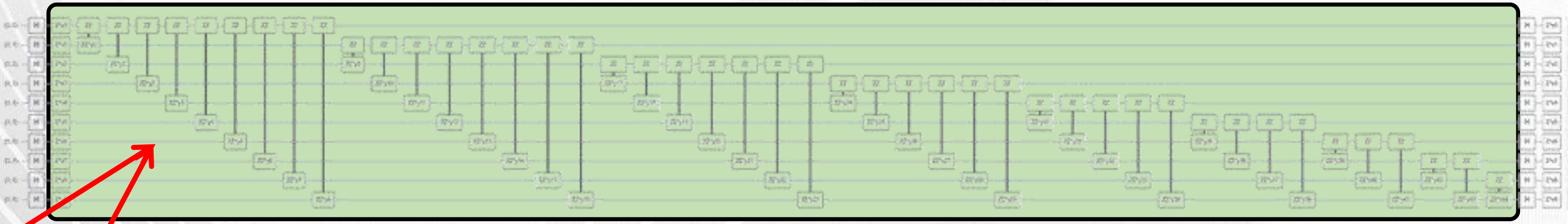
QAI on Rainfall radar images – Preprocessing



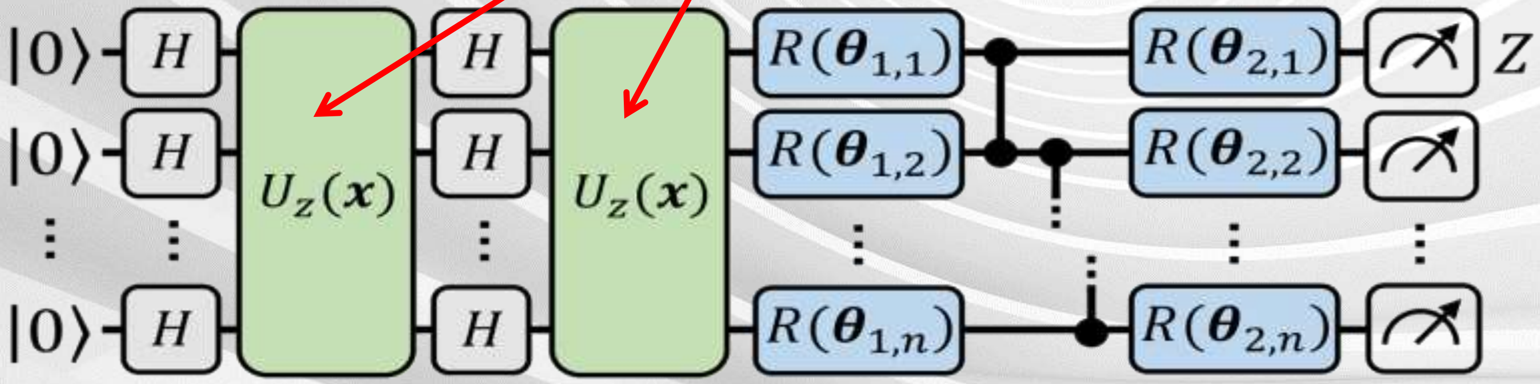
data source:
maps.dwd.de/geoserver

6125 * 7250 pixels per image
14 (plus 1) timesteps; 2 "colors" each
7 * 4 = 28 pixels per cutout
~ 1350 * 2250 cutouts
with 28 * (2 * 14) "infodots"
~ 3.000.000 input datasets
~ 8 datasets each km²

Quantum Machine Learning Beyond Kernel Methods – Algorhythm

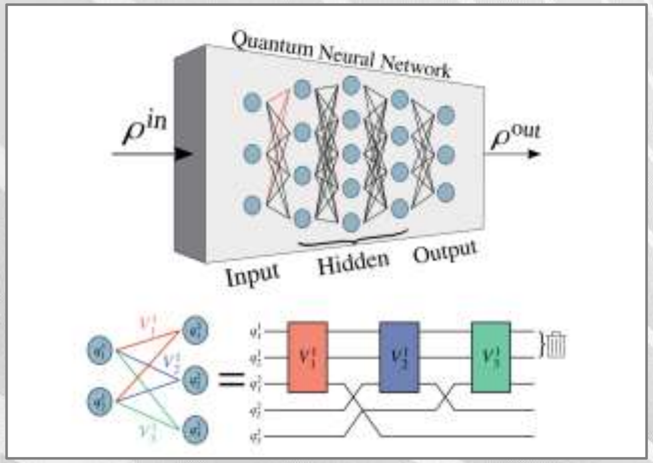


source:
Sofiene Jerbi et.al.
QML beyond kernel methods; 02/2022

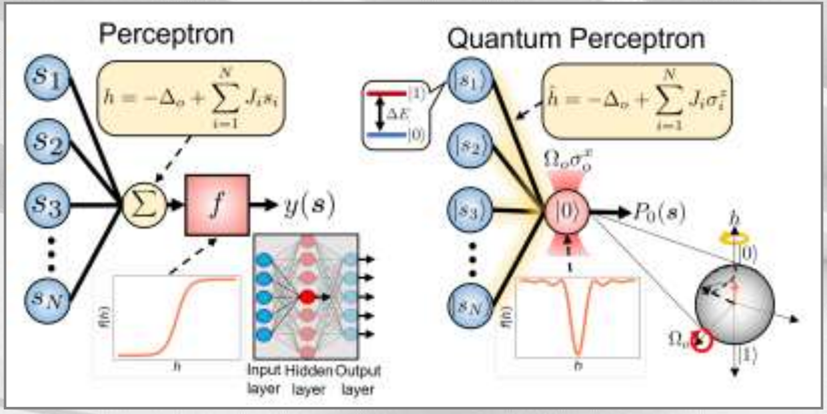


The model uses the feature encoding proposed by Havlíček et al., followed by a hardware-efficient variational circuit, where arbitrary single-qubit rotations on each qubit are interlaid with nearest-neighbour gates, for L layers. Finally, the expectation value of a Z observable assigns labels to input data.

Dissipative Perceptron-based Quantum Neural Network (DPBQNN)

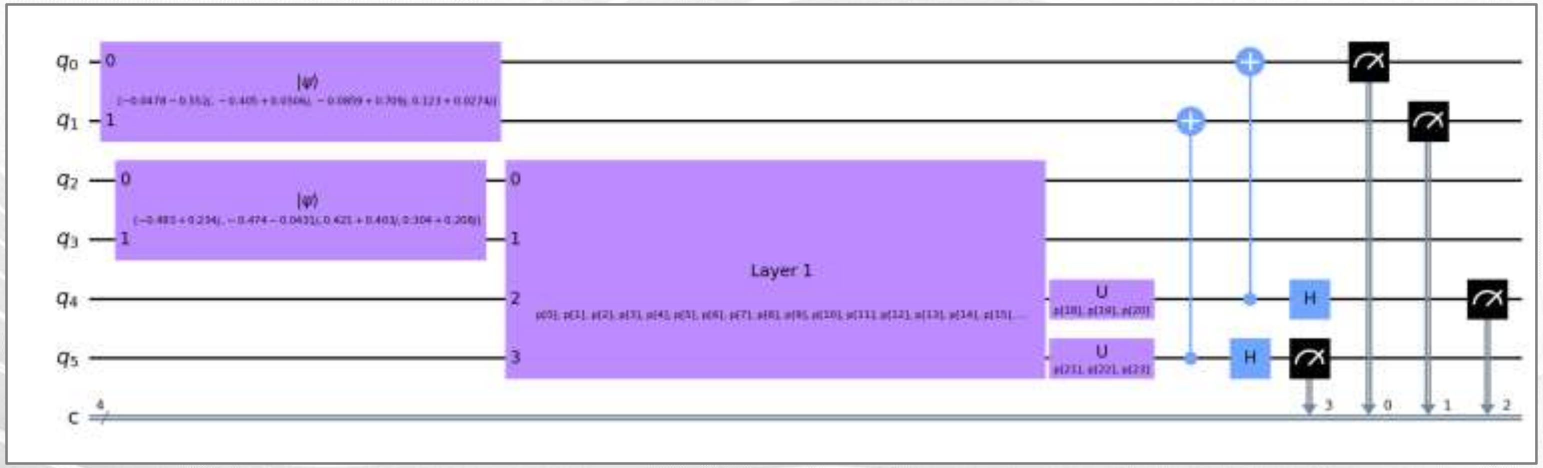


source: Kerstin Beer
Quantum neural networks, 05/2022

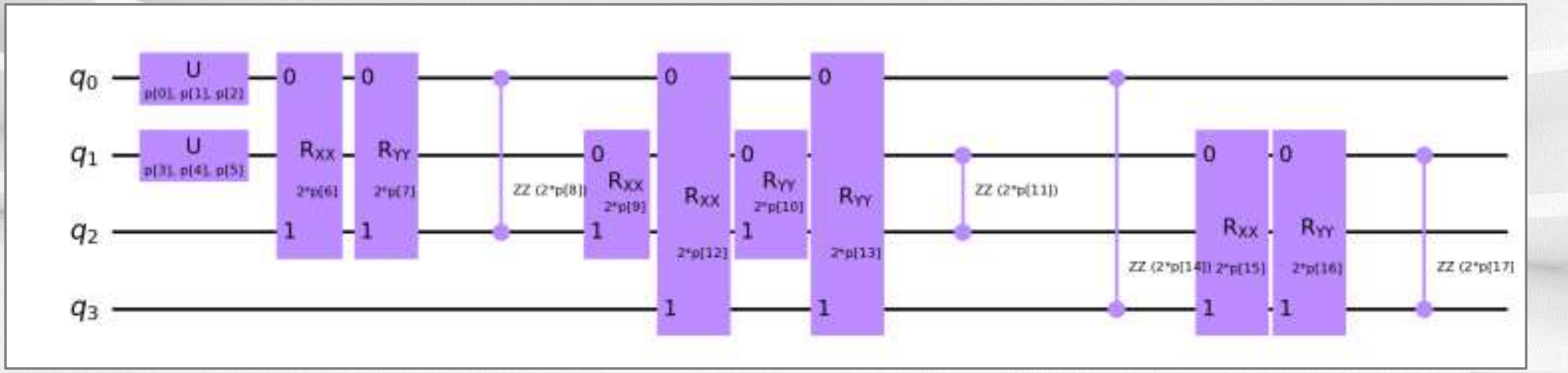


source: Rodrigo Araiza Bravo et.al.
Universal Quantum Perceptrons for QML, 12/2022

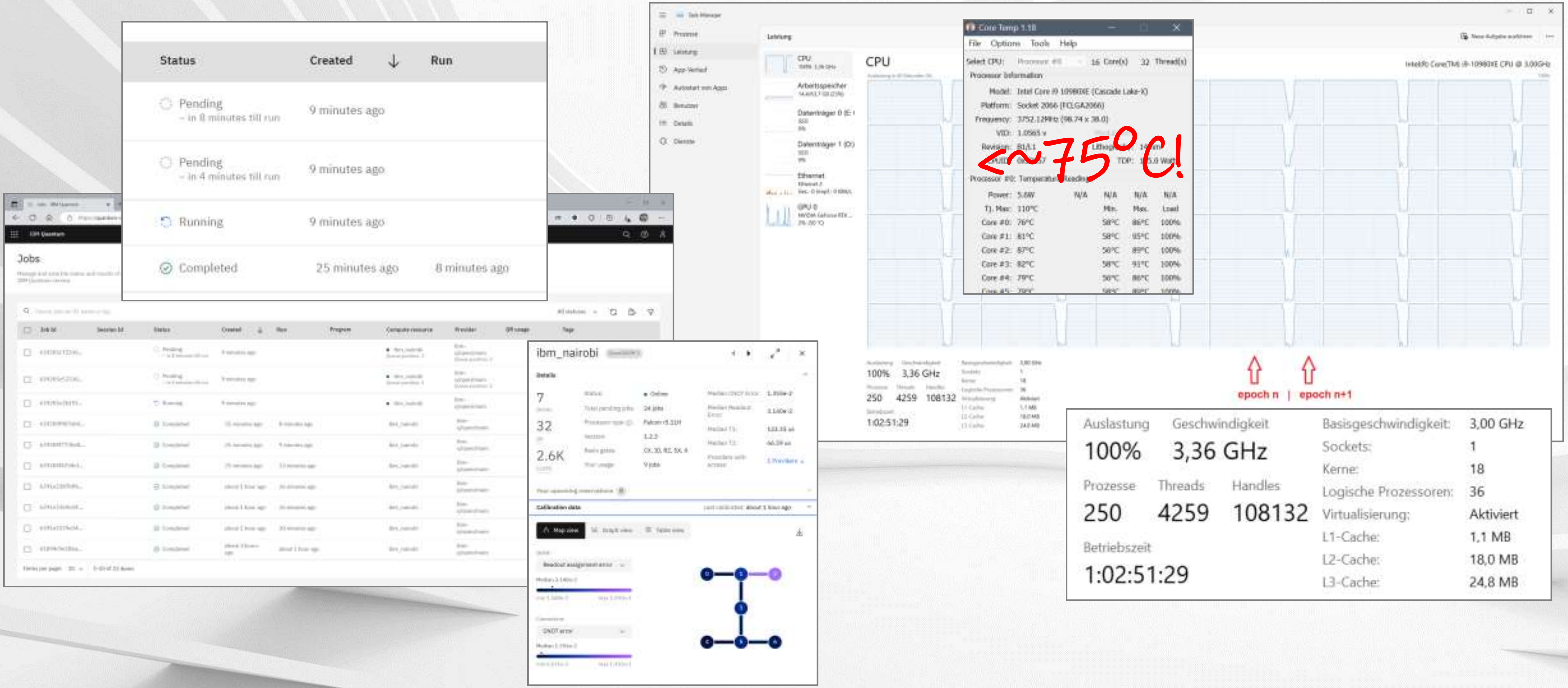
circuit



circuit – unitary layer1



DPBQNN – Quantum Hardware vs. Quantum Simulation



Status	Created	Run
Pending – in 8 minutes till run	9 minutes ago	
Pending – in 4 minutes till run	9 minutes ago	
Running	9 minutes ago	
Completed	25 minutes ago	8 minutes ago

Select CPU: 16 Core(s) 32 Thread(s)

Processor Information

Model: Intel Core i9 10980XE (Cascade Lake-X)
Platform: Socket 2066 (FCLGA2066)
Frequency: 3752.12MHz (98.74 x 38.0)
VID: 1.0565 v
Revision: B1/L1

Processor #0: Temperature Readings

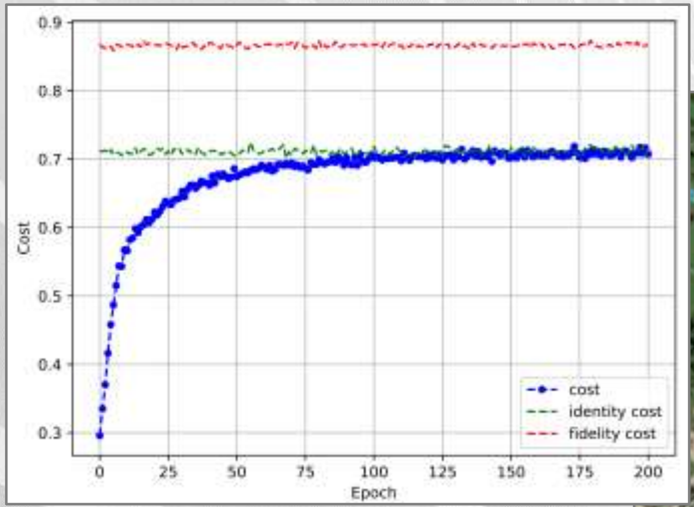
			Min.	Max.	Load
Power:	5.6W	N/A	N/A	N/A	N/A
Tj. Max:	110°C				
Core #0:	76°C		58°C	86°C	100%
Core #1:	81°C		58°C	95°C	100%
Core #2:	87°C		56°C	89°C	100%
Core #3:	82°C		58°C	91°C	100%
Core #4:	79°C		56°C	86°C	100%

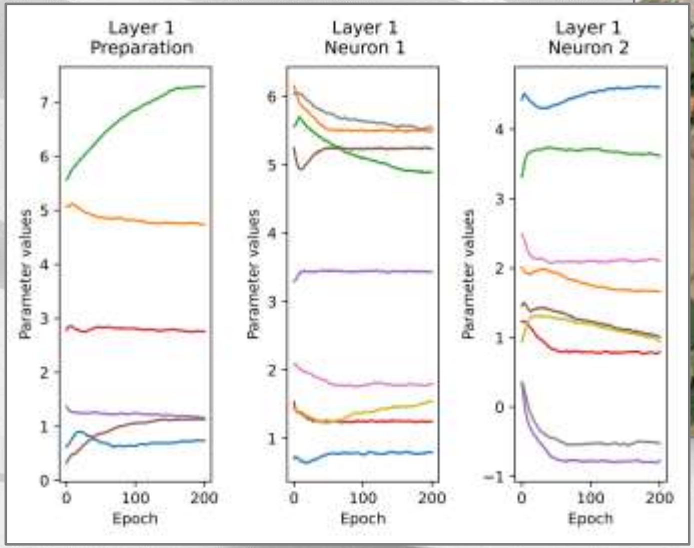
<~75°C!

epoch n | epoch n+1

Auslastung	Geschwindigkeit		Basisgeschwindigkeit:	3,00 GHz
100%	3,36 GHz		Sockets:	1
Prozesse	Threads	Handles	Kerne:	18
250	4259	108132	Logische Prozessoren:	36
			Virtualisierung:	Aktiviert
Betriebszeit			L1-Cache:	1,1 MB
1:02:51:29			L2-Cache:	18,0 MB
			L3-Cache:	24,8 MB

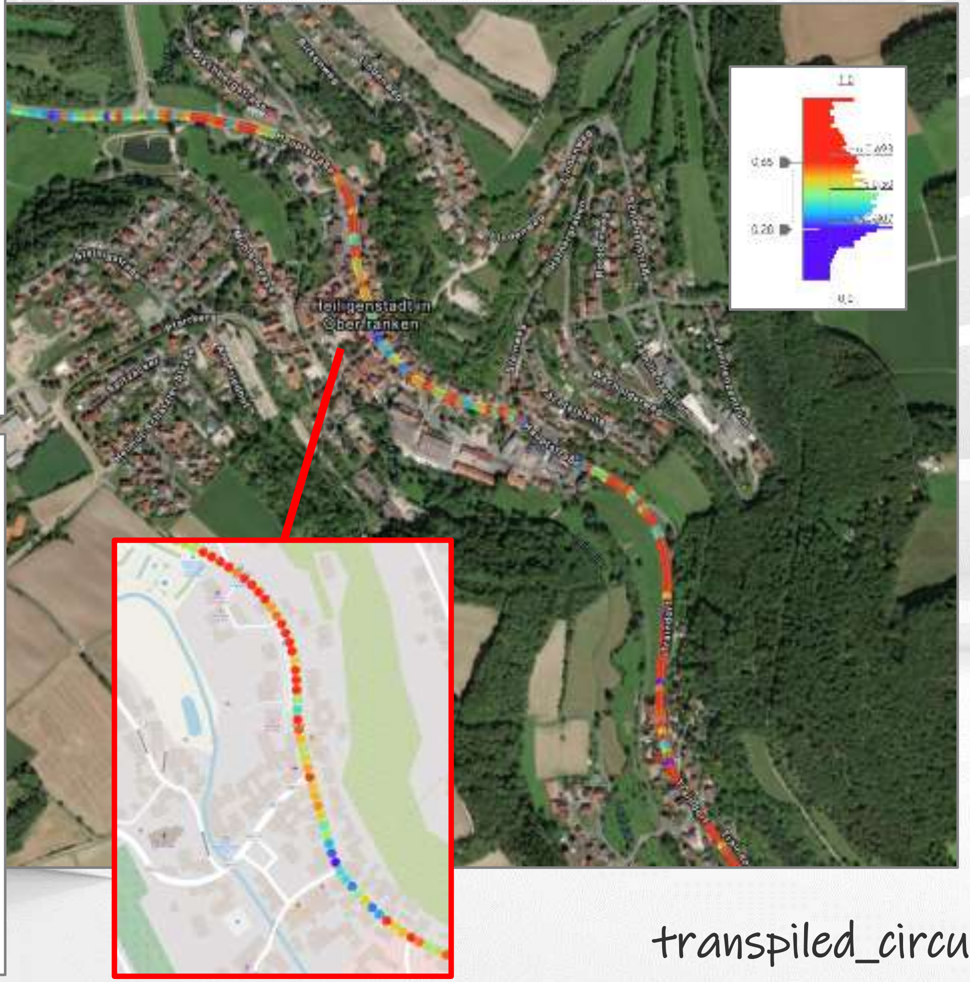
Predictive Maintenance using DPBQNN

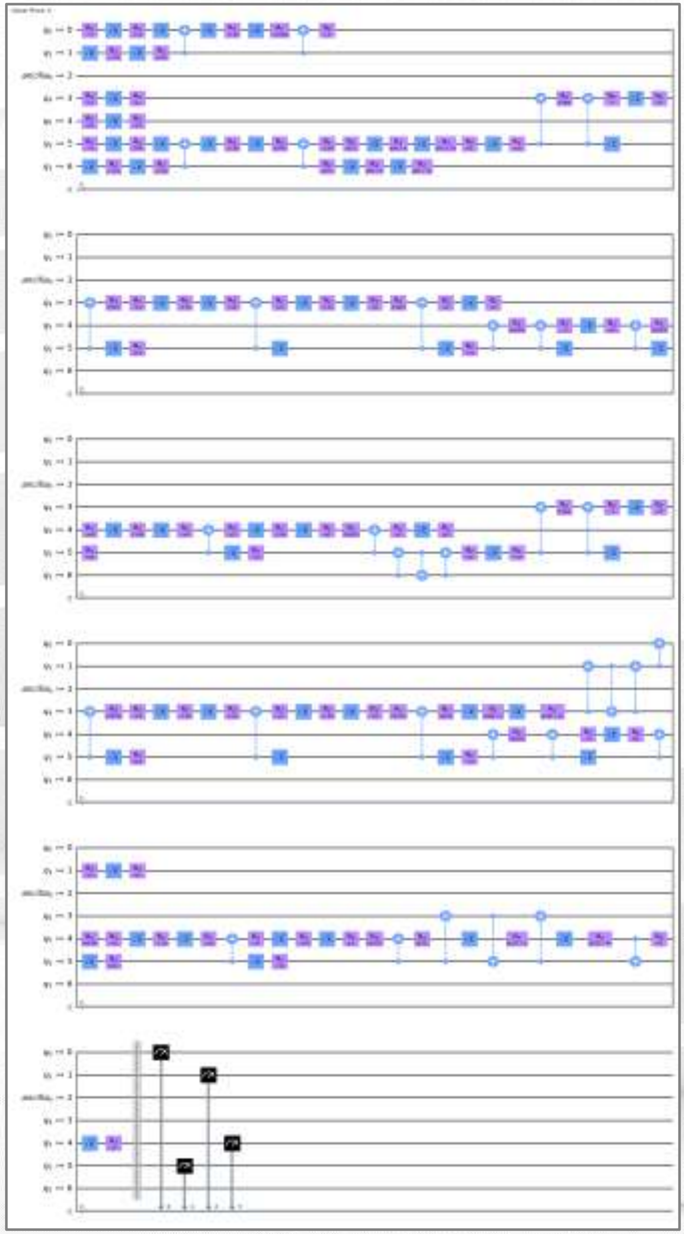




data source:
www.stmb.bayern.de

used hardware: ibm_nairobi





transpiled_circuit

An Executive Checklist for Quantum Artificial Intelligence

- **Create a strategy**
  Establish a framework for how Quantum AI will be incorporated into business strategy and processes, and to define measurable goals.
- **Apply executive support**
  Assign a C-level executive to oversee the company's strategy.
- **Incorporate robust datasets, including location information**
  Business data can become more valuable when coupled with information about its location and time.
- **Mind the data**
  Predictions will be accurate only if the training data is truly representative of the target cases.