Hands-On Workshop on Quantum Machine Learning

Mohammad Saidur Rahman Ph.D.
Jesus R. Lopez Viviana Cadena

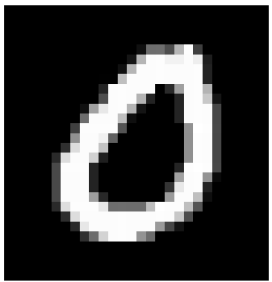
https://rahmanmsaidur.com/



UTEP Computer Science

Can a computer recognize this handwritten digit?

ONE REAL MNIST IMAGE



What digit is this?

Answer from the dataset: 0

The image is a real MNIST test example. We will follow it to a predicted label.

From handwritten image to quantum computation

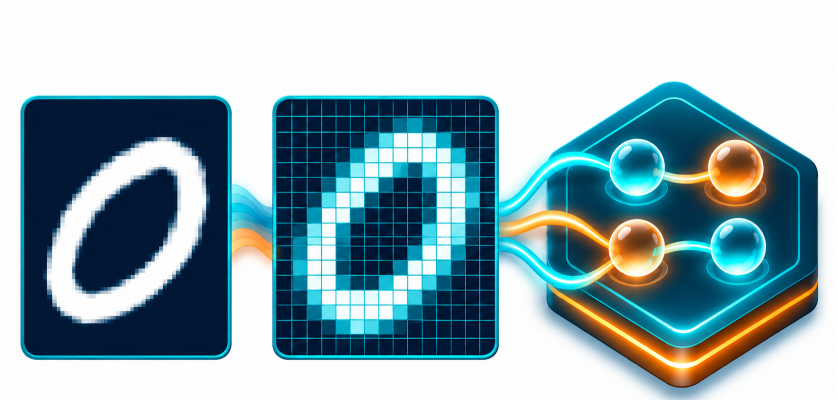


Illustration of the journey. Later diagrams show the actual four-qubit implementation.

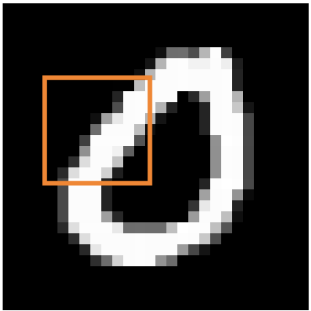
The question we will answer



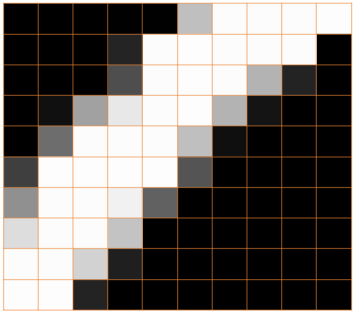
Our task is binary: distinguish handwritten 0 from 1. We will implement each stage ourselves.

An image is a grid of numbers

28 x 28 pixels

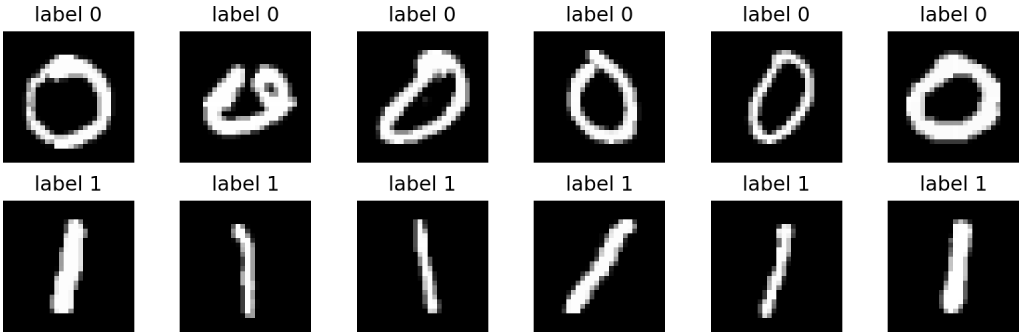


Each square is one number



The 28 by 28 image is 784 pixel intensities. Its label is the known answer during training.

Labeled examples show the task



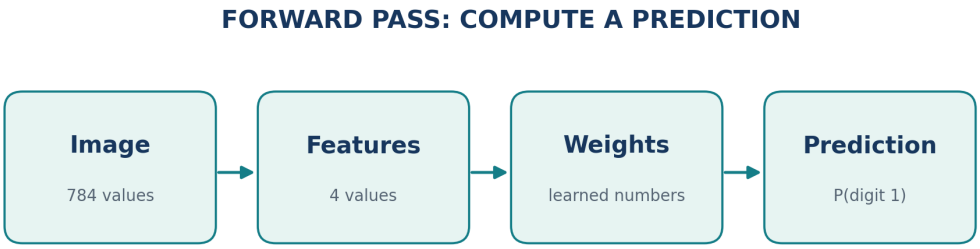
Actual examples from the workshop subset. The model learns from many images, not one rule.

Three sets, three different jobs

TRAIN	VALIDATION	TEST
160 images	40 images	200 images
Learn weights and fit PCA	Choose the checkpoint	Report final performance

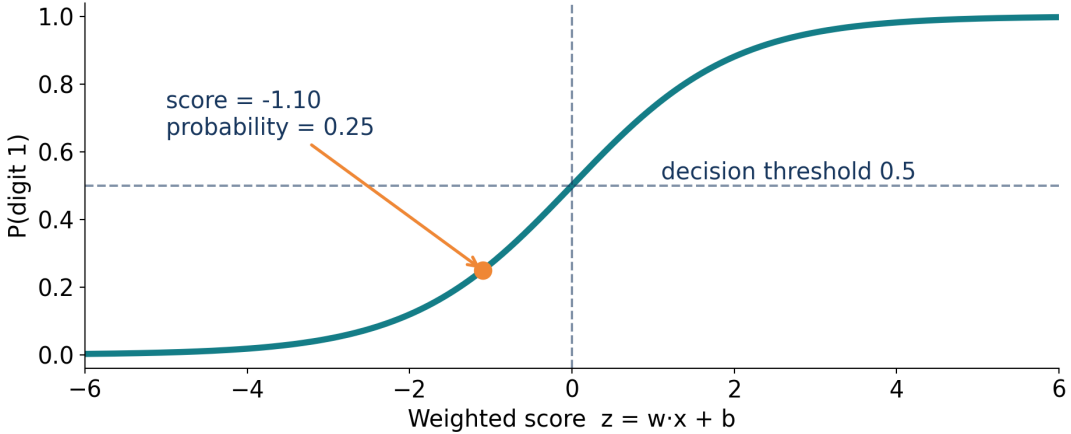
All three sets are balanced between 0 and 1. The test images stay unseen until the model is frozen.

The classical forward pass



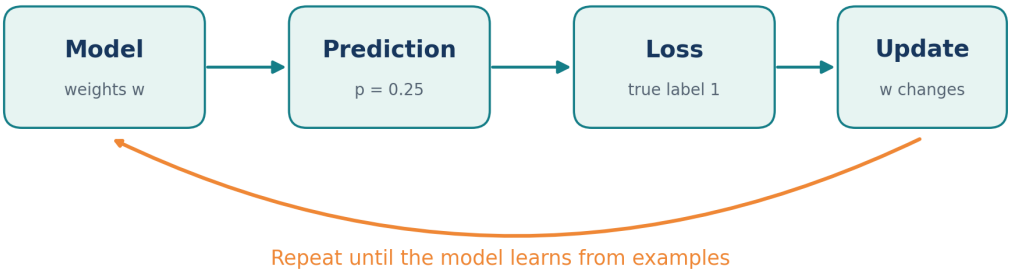
Four features enter a weighted function. Its output is a probability for digit 1.

The classical model turns a score into a probability



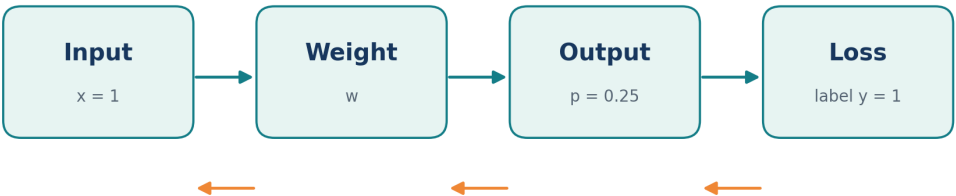
Logistic regression learns four weights and an intercept. A weighted score of -1.10 gives $P(1) = 0.25$.

The classical learning loop



Predictions and known labels produce a loss. Repeated updates change the model weights.

What backpropagation computes



BACKWARD: sensitivity of loss to weight = (p - y)x = -0.75

The backward calculation tells us how a small change in each weight would change the loss.

One numerical weight update

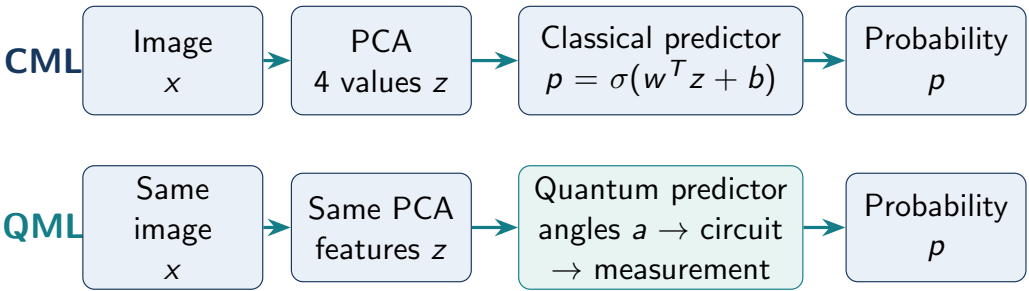
Input $x = 1$ Correct label $y = 1$ Current $p = 0.25$

$$\frac{\partial L}{\partial w} = (p - y)x = -0.75$$



Increasing w raises the probability of the correct digit. Backpropagation obtains the derivative; the optimizer performs the update.

The QML pipeline changes the predictor



During training, both compare p with the known label y and use a classical optimizer. CML learns weights w, b; QML learns circuit angles θ. Both use the same held-out test images.

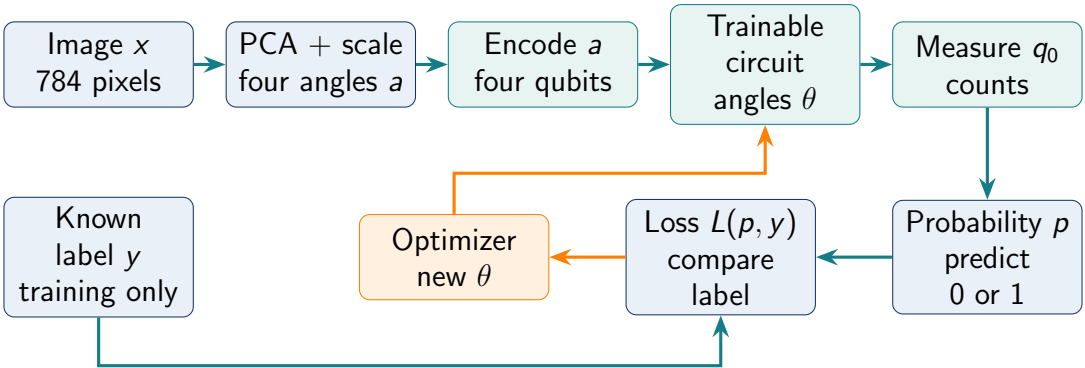
QML in one sentence

A quantum circuit computes a prediction from encoded data.

A classical computer prepares images, compares predictions with labels, and chooses new circuit angles.

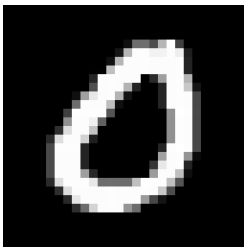
This is one kind of QML: a hybrid variational quantum classifier. The term QML also includes quantum kernels and other methods.

The complete hybrid classifier



Forward: image → probability. Training: label → loss → updated circuit angles. At inference, freeze θ and omit the feedback path.

One image: pixels become gate angles

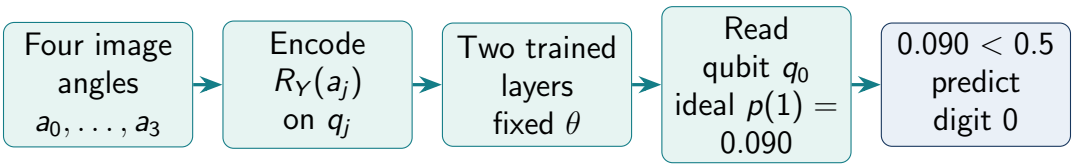


Real test image x

1. **Pixels**: 28 by 28 becomes 784 normalized values.
2. **PCA**: a training-fitted transform gives four features. $z = (-5.67,\ 0.37,\ 1.64,\ 1.31)$.
3. **Scale**: a training-fitted scaler gives gate angles. $a = (0.72,\ 1.70,\ 2.02,\ 1.97)$ radians.

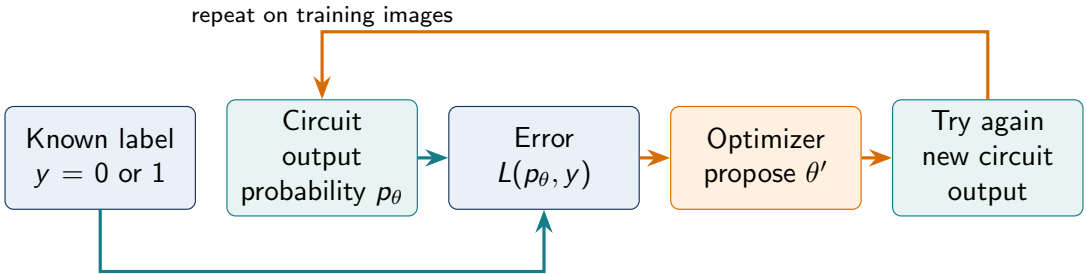
Each a_j depends on this image and controls one R_Y gate. The learned angles θ are fixed when this test image is classified.

One image: the circuit makes a prediction



The known label is 0, so this prediction is correct. The label checks the answer; it does not enter the prediction circuit. Here 0.090 is an ideal simulator result; hardware estimates p from repeated shots.

The training feedback loop changes the model



Loss is calculated on labeled training images. The optimizer proposes the next angles. Validation chooses a checkpoint; the test set is used only after freezing it.

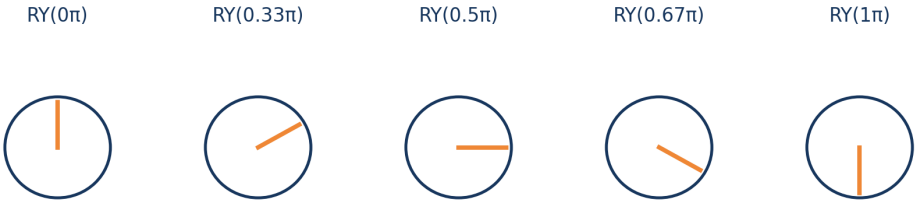
Check your understanding

For a new image, which values change?

1. The four encoding angles a
2. The trained gate angles θ
3. The correct label y fed into the prediction

Answer: only a changes with a new image. The trained θ remains fixed, and the label is not fed into the prediction.

An angle changes a qubit probability

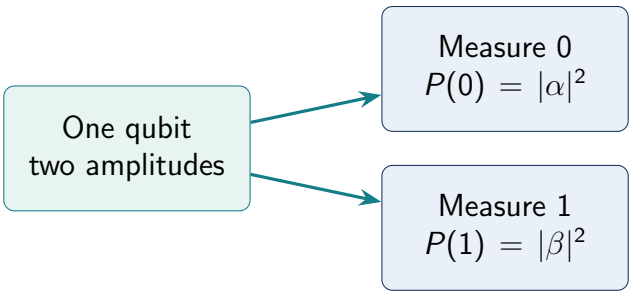


The angle changes what repeated measurements will report

$R_Y(a)|0\rangle$ has measurement probability $P(1) = \sin^2(a/2)$. The angle controls the chance of 1.

A qubit has amplitudes and measurable probabilities

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$



One shot gives one classical outcome. Repeated preparations estimate the two probabilities; $|\alpha|^2 + |\beta|^2 = 1$.

Hadamard makes phase matter

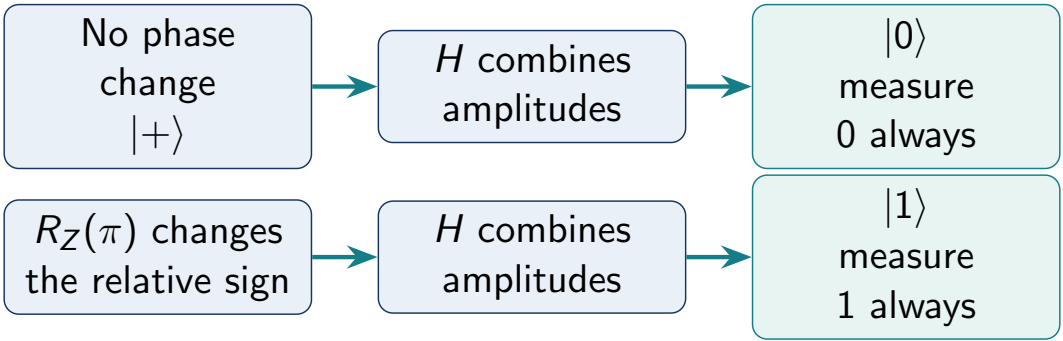
$$|0\rangle \xrightarrow{H} |+\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}} \xrightarrow{H} |0\rangle$$

$$|1\rangle \xrightarrow{H} |-\rangle = \frac{|0\rangle-|1\rangle}{\sqrt{2}} \xrightarrow{H} |1\rangle$$

The two middle states both give 50/50 when measured immediately. Their different relative phase changes what a later H produces.

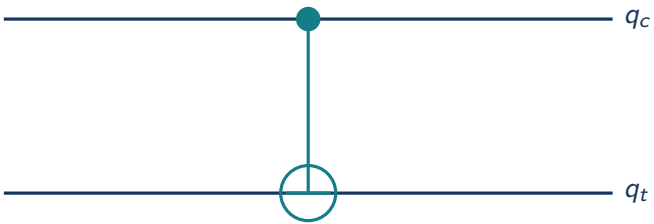
Phase is hidden until another gate reveals it

Both paths start with $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$.



Before H, both paths give 50/50. R_Z changes relative phase; H turns it into a measurable difference. Overall global phase has no effect.

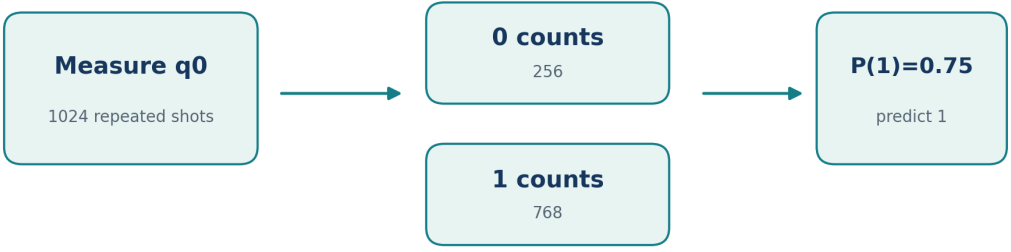
CNOT links two qubits



$$|c, t\rangle \mapsto |c, t \oplus c\rangle$$

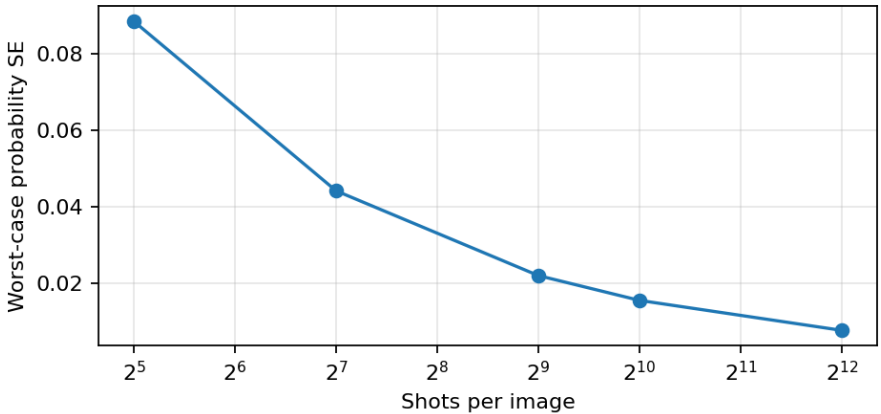
The upper qubit is the control. The lower target flips only when the control is 1. In our model, a chain of CNOTs lets input wires affect the measured wire.

Measurement requires fresh preparations



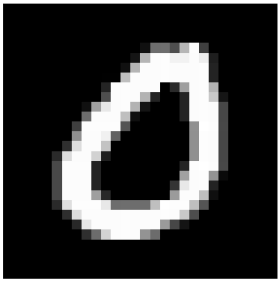
One shot prepares the circuit and measures once. Repeating it gives counts, then a probability estimate.

More shots reduce sampling noise



Repeating the circuit estimates a probability; finite shots fluctuate even without device errors.

The same image becomes four numbers



PCA fitted on training images

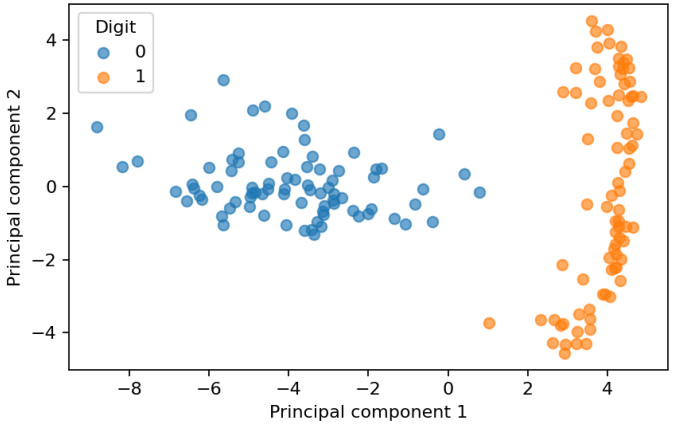
FOUR FEATURES

-5.67, 0.37, 1.64, 1.31

Four rotation angles: 0.72, 1.70, 2.02, 1.97 rad

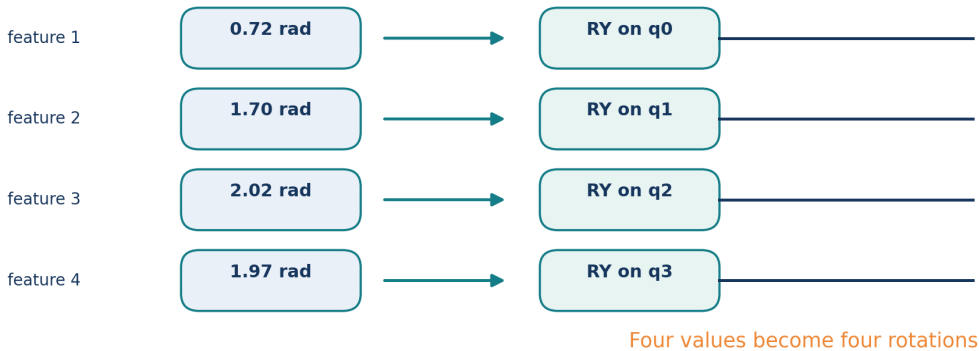
PCA maps 784 pixels to four features. A training-fitted scaler converts them to rotation angles.

PCA reveals a compact representation



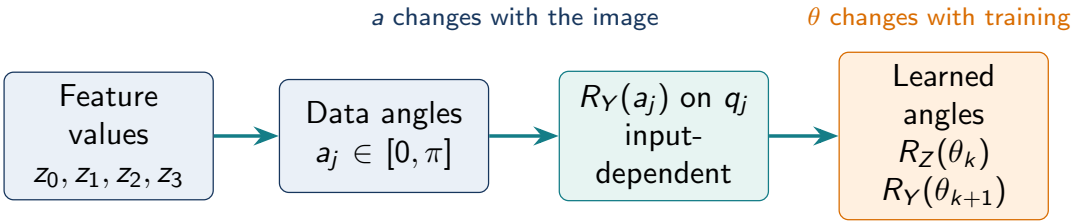
The four-component PCA transform is fitted on training images. It is then reused on validation and test.

Data encoding on four qubits



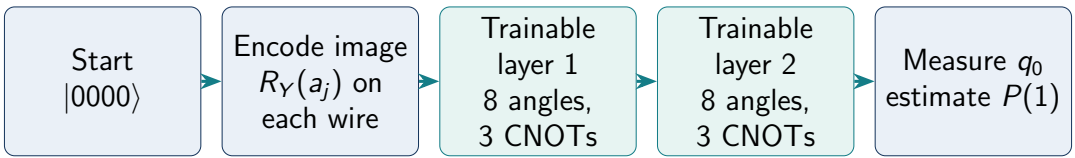
Each $R_Y(a_j)$ uses an image-dependent angle. The learned θ values come later.

Part 3: data encoding is separate from learning



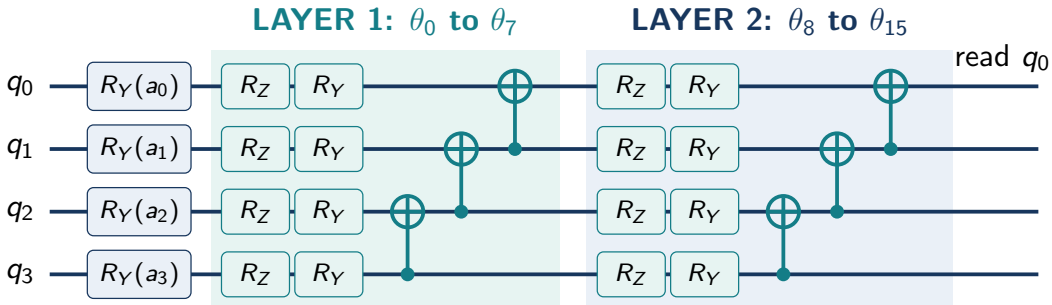
Angle encoding maps four classical numbers to four rotation gates. Encoding values carry the image; trainable angles define the classifier.

Inside the predictor: encode, two layers, measure



IBM calls the encoding circuit a *feature map* and the trainable circuit an *ansatz*. Our feature map uses four data angles a; the two ansatz layers use 16 learned angles θ.

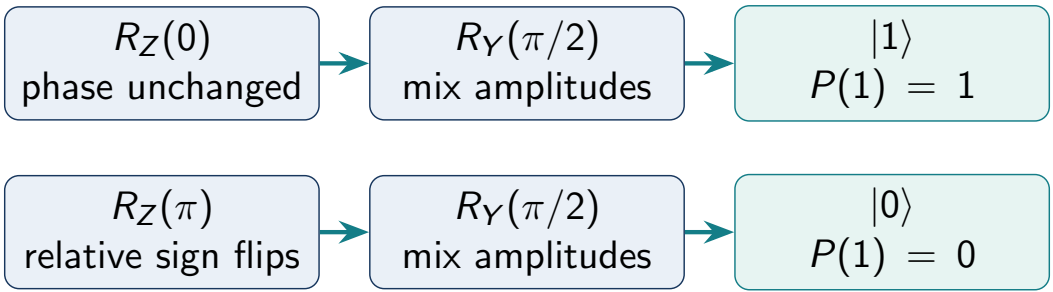
Layer 1 and layer 2 are repeated gate patterns



On every wire, each shaded layer has R_Z then R_Y, followed by CNOTs $q_3 \to q_2$, $q_2 \to q_1$, $q_1 \to q_0$. The two layers repeat the pattern with different learned angles.

Why the order RZ then RY matters

Start with $|+\rangle$. Either R_Z choice alone still measures 0 or 1 with 50/50 probability.



The later R_Y turns the phase difference into different outcomes. This isolates the gate order; the classifier also uses CNOTs and two layers.

How repeated measurements become an answer



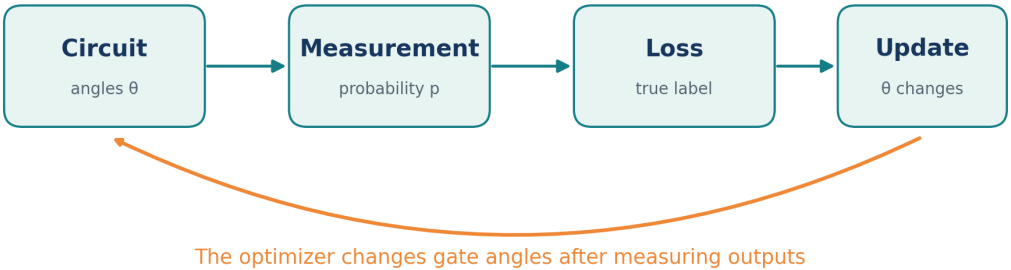
Illustrative count example: 768 ones in 1024 shots give $p = 0.75$, so the threshold predicts 1.

Turning the quantum state into a binary prediction

$$p_\theta(y = 1 \mid a) = P(q_0 = 1) = \frac{1 - \langle Z_0 \rangle}{2}$$

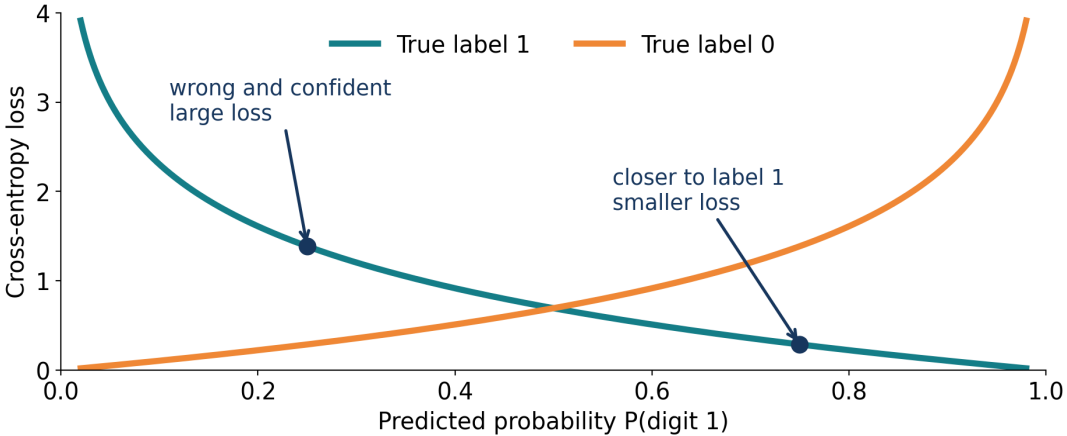
- Outcome 0 has Z eigenvalue $+1$. Outcome 1 has Z eigenvalue -1.
- For probabilities 0.25 and 0.75, the Z expectation is -0.5.
- A fixed threshold of 0.5 converts the measured probability to a digit label.
- A measurement probability is not automatically a calibrated confidence score.

The hybrid training loop



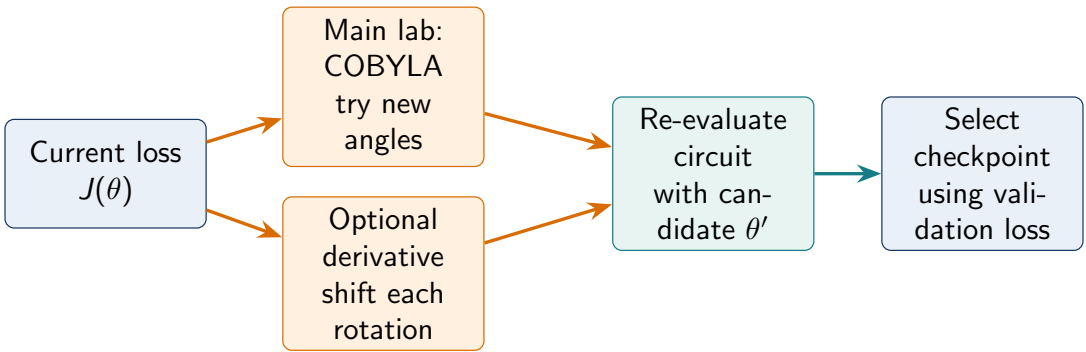
The quantum circuit supplies p_θ. A classical loss and optimizer update θ.

Loss penalizes confident mistakes



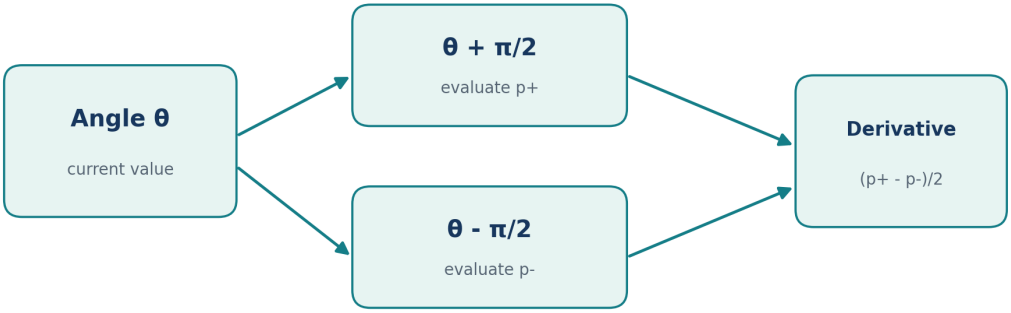
For a true 1, $p = 0.25$ has loss 1.386; $p = 0.75$ has loss 0.288. The model minimizes average loss.

Two ways to choose new circuit angles



COBYLA does not request gradients. Parameter shift can obtain circuit derivatives from shifted evaluations, then the classical chain rule connects them to the loss.

If we want a derivative



Two shifted evaluations can give the probability derivative for a supported rotation.

Circuit derivative and loss derivative are different

Step 1: estimate how a gate angle changes the measured probability

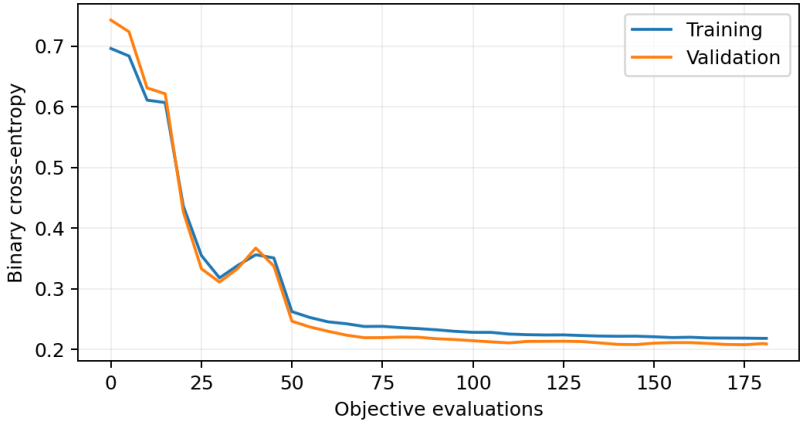
$$\frac{\partial p}{\partial \theta_j} = \frac{p(\theta_j + \pi/2) - p(\theta_j - \pi/2)}{2}$$

Step 2: connect that change to the training loss

$$\frac{\partial L}{\partial \theta_j} = \frac{\partial L}{\partial p}\frac{\partial p}{\partial \theta_j}$$

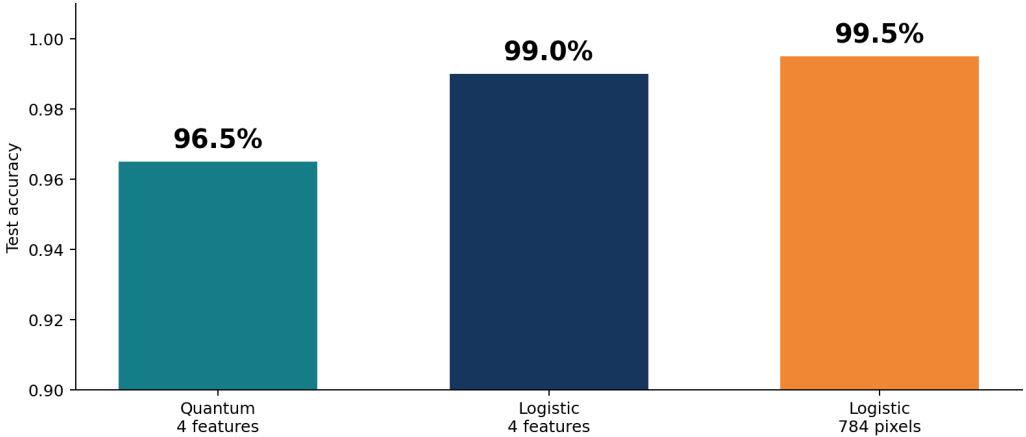
The main notebook trains with COBYLA and does not require either derivative. The two-step calculation explains gradient-based alternatives.

The local training curve



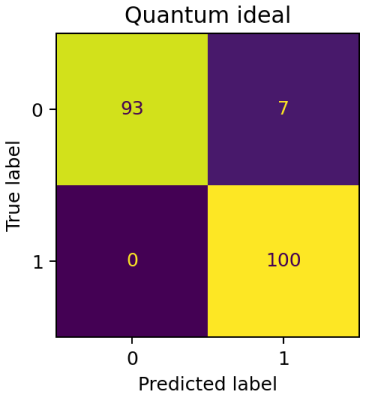
Training and validation losses fall in the saved local run. We select a checkpoint before the test.

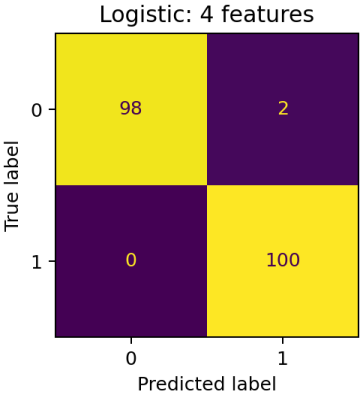
Test accuracy against classical baselines



On this small held-out subset, the classical baselines outperform the ideal quantum model.

Which digits were misclassified?





The quantum model classified 193 of 200 correctly. The four-feature logistic baseline classified 198.

Verified local results

Model	Test accuracy	Cross-entropy
Quantum ideal	96.5%	0.228
Logistic 4 features	99.0%	0.059
Logistic 784 pixels	99.5%	0.023

All results use the fixed 200-image test subset. Quantum results are ideal local simulation, not IBM hardware. Four PCA components retain 55.3% of training variance.

The classical model wins this comparison. A useful workshop does not require quantum superiority.

The notebook follows the same path



Prepare data, encode it, transform it with gates, read a probability, and train.

Notebook laboratory: leakage-free preprocessing

```
pca = PCA(n_components=4, svd_solver='full')
z_train = pca.fit_transform(flat_train)
z_val = pca.transform(flat_val)
z_test = pca.transform(flat_test)
scaler = MinMaxScaler((0, np.pi), clip=True)
A_train = scaler.fit_transform(z_train)
A_val = scaler.transform(z_val)
A_test = scaler.transform(z_test)
```

Google Colab

Jupyter Notebook file is available at
https://tinyurl.com/QML-UTEP

Construct the feature and trainable blocks

```
features = ParameterVector('x', 4)
weights = ParameterVector('theta', 16)
circuit = QuantumCircuit(4)
for q in range(4):
    circuit.ry(features[q], q)
k = 0
for layer in range(2):
    for q in range(4):
        circuit.rz(weights[k], q)
        circuit.ry(weights[k+1], q)
        k += 2
    for q in (3, 2, 1):
        circuit.cx(q, q-1)
```

Notebook laboratory: fitting the circuit

```
fit = minimize(objective, theta_initial,
    method='COBYLA',
    options={'maxiter': 180, 'rhobeg': 0.6})
# objective evaluates training loss and records
# validation loss every five evaluations.
theta_best = best['theta'].copy()
```

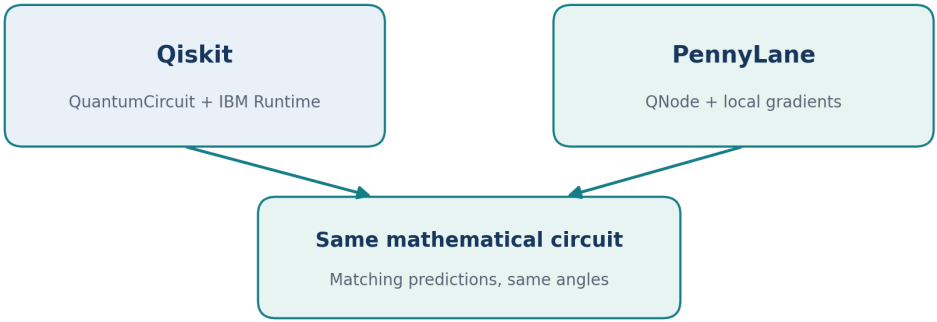
The selected checkpoint can differ from the final optimizer point. Test data never selects it.

Finite-shot prediction uses measured circuits

```
result = StatevectorSampler(seed=17).run(
    measured, shots=1024).result()
counts = result[0].data.readout.get_counts()
p_one = counts.get('1', 0) / sum(counts.values())
```

The same named-register extraction pattern is used for IBM Runtime results.

One circuit expressed in two libraries



Qiskit handles the IBM submission. PennyLane makes the equivalent circuit and local differentiation visible.

Qiskit and PennyLane have different roles

	Qiskit	PennyLane
Today	Main circuit, simulator, hardware jobs	Equivalent teaching circuit and automatic differentiation
Typical object	QuantumCircuit	QNode
Local simulation	StatevectorSampler or Statevector	default.qubit
IBM run here	IBM Runtime SamplerV2	Not used for the submitted job

The two libraries are demonstrated side by side. The hardware route in this package uses Qiskit.

The same four-qubit classifier in PennyLane

```
dev = qml.device('default.qubit', wires=4)
@qml.qnode(dev, interface='autograd',
           diff_method='parameter-shift')
def qnode(x, theta):
    for q in range(4): qml.RY(x[q], wires=q)
    for layer in range(2):
        for q in range(4):
            k = 8*layer + 2*q
            qml.RZ(theta[k], wires=q)
            qml.RY(theta[k+1], wires=q)
        for q in (3, 2, 1): qml.CNOT(wires=[q,q-1])
    return qml.expval(qml.PauliZ(0))
```

The notebook verifies $p(1) = (1 - \langle Z_0 \rangle)/2$ against Qiskit's exact output.

Automatic differentiation still executes circuits

```
t = pnp.array(theta_best, requires_grad=True)
def probability(theta):
    return (1 - qnode(A_val[0], theta)) / 2
sensitivity = qml.grad(probability)(t)
```

PennyLane connects the QNode to the chain rule. The chosen differentiation method determines the circuit-evaluation cost. Automatic differentiation does not remove shot noise or hardware queueing.

A quantum computer is physical laboratory equipment

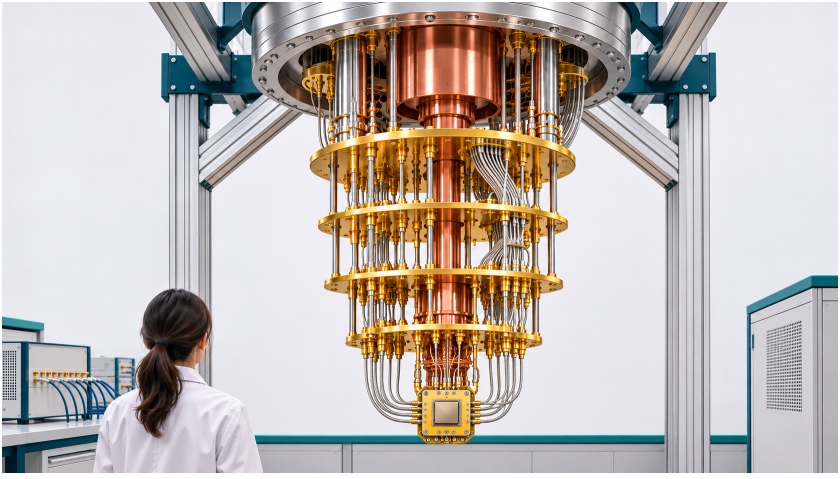
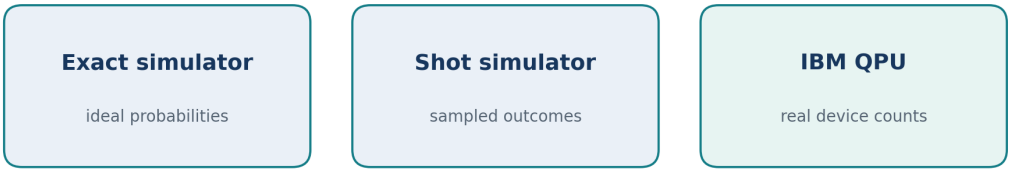


Illustration of a dilution refrigerator. The IBM backend is selected when the notebook runs.

Exact, sampled, and device results differ



Same frozen circuit and images, different execution conditions

Only the physical device adds actual gate and readout errors; all modes use the frozen model.

The IBM job flow



Authenticate, select a backend, transpile, submit once, save the job ID, and retrieve counts.

Transpilation connects a logical model to a device

- Convert abstract operations into the target instruction set.
- Map logical qubits to physical qubits.
- Route two-qubit operations through available connectivity.
- Inspect compiled depth, two-qubit gate count, and layout.
- Compile against the same backend used for submission.

The live inference budget

Quantity	Default
Preselected test images	12
Bound circuits per image	1
Shots per circuit	1024
Total circuit shots	12,288
Runtime submissions	1

This is a workload count, not a price quote or a wall-clock estimate. Review your current quota and service limits.

Submit and record the job ID

```
sampler = SamplerV2(mode=backend)
sampler.options.max_execution_time = 300
job = sampler.run(isa_circuits, shots=1024)
job_id = job.job_id()
print(job_id)
# Save the ID, backend, indices, and checkpoint
# immediately, before waiting for job.result().
```

The notebook prevents accidental duplicate submission by reusing its saved job ID.

Recover a job after a notebook disconnects

```
job = service.job(saved_job_id)
print(job.status())
result = job.result()
counts = [r.data.readout.get_counts()
          for r in result]
p_hardware = np.array([
    c.get('1', 0) / sum(c.values())
    for c in counts])
```

Waiting can take longer than the teaching block. Continue exercises and retrieve later rather than submitting duplicates.

The hardware report must identify its scope

- Record the job ID, backend, selected image indices, shot counts, and timestamp/metadata.
- Report 12-image hardware accuracy separately from 200-image ideal test accuracy.
- Plot ideal and measured probabilities for the same selected images.
- Keep raw counts so later analysis can reproduce the estimates.
- The delivered package contains no fabricated IBM hardware measurements.

What the workshop establishes

- A complete, inspectable path from MNIST pixels to quantum measurement probabilities.
- A trained variational circuit with a classical baseline and held-out evaluation.
- A concrete IBM Runtime workflow with target compilation and recoverable jobs.
- An optional demonstration of measurement-driven hardware parameter updates.
- A foundation for larger experiments, with explicit limits on scientific claims.

Exercise set A: foundational reasoning

- Predict $P(1)$ after RY(0), RY($\pi/2$), and RY(π).
- Explain why H followed by H returns the original basis state.
- How many shots halve the independent-sampling standard error?
- Why does one four-qubit shot not reveal all sixteen amplitudes?
- Where is the classical component in this hybrid quantum classifier?

Exercise set A: answers

- The three probabilities are 0, 0.5, and 1.
- Coherent amplitudes interfere before measurement and $H^2 = I$.
- Four times as many shots.
- A shot yields one sampled bitstring, not the statevector.
- Preprocessing, loss computation, parameter updates, and result analysis are classical.

Read these first

- Biamonte et al. (2017), *Quantum machine learning*, Nature 549, 195–202. https://doi.org/10.1038/nature23474
- Mitarai et al. (2018), *Quantum Circuit Learning*, Physical Review A 98, 032309. https://doi.org/10.1103/PhysRevA.98.032309
- Schuld and Killoran (2019), *Quantum machine learning in feature Hilbert spaces*, Physical Review Letters 122, 040504. https://doi.org/10.1103/PhysRevLett.122.040504
- Havlíček et al. (2019), *Supervised learning with quantum-enhanced feature spaces*, Nature 567, 209–212. https://doi.org/10.1038/s41586-019-0980-2
- Schuld et al. (2020), *Circuit-centric quantum classifiers*, Physical Review A 101, 032308. https://doi.org/10.1103/PhysRevA.101.032308

What participants will be able to do

- Explain qubits, amplitudes, interference, entanglement, and measurement.
- Turn MNIST images into features and encode them in a quantum circuit.
- Train a variational classifier and compare it with a classical baseline.
- Transpile, submit, retrieve, and interpret an IBM quantum hardware job.
- Distinguish a working demonstration from evidence of quantum advantage.

Suggested teaching schedule

Block	Content	Minutes
Foundations	ML, qubits, gates, measurement	45
Model design	MNIST, PCA, encoding, readout	35
Local laboratory	Training, metrics, shot noise	45
IBM laboratory	Accounts, transpilation, job retrieval	35
Advanced topics	Hardware updates, kernels, ten digits	30
Exercises	Discussion and troubleshooting	20

Submit the hardware inference job early in the IBM block, then discuss noise while it queues.

Supervised learning starts with labeled examples

- An input x is an image. A target y is its known digit label.
- The model learns parameters from examples rather than receiving handwritten recognition rules.
- A binary classifier estimates $p(y = 1 \mid x)$. A threshold converts the probability into a label.
- Our threshold is fixed at 0.5. Predict 1 at or above it, otherwise 0.

Metrics answer different questions

Metric	Meaning
Accuracy	Fraction of predicted labels that are correct
Balanced accuracy	Mean of recall for digit 0 and digit 1
Cross-entropy	Quality of predicted probabilities against true labels
Confusion matrix	Counts of correct and incorrect predictions for each class

A 50% majority-class reference applies to our balanced binary subset. It is not the ten-class baseline.

A plain-language definition

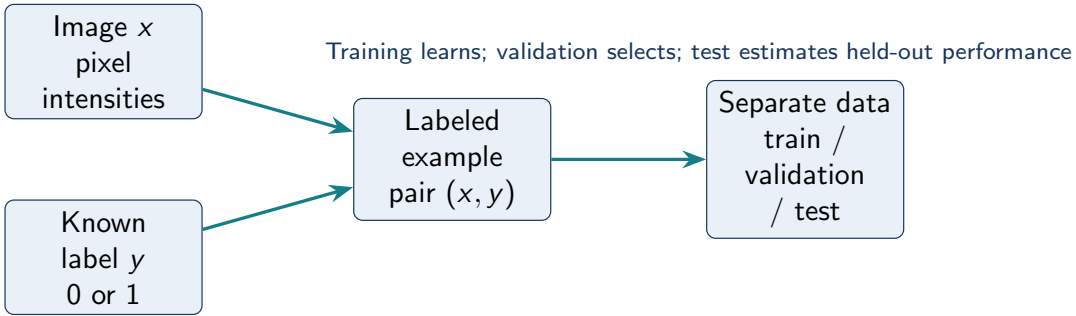
- Quantum machine learning studies learning methods that use quantum information processing in a meaningful part of the model or data.
- Today: a quantum circuit supplies the probability for a digit. A classical computer chooses its gate angles.
- Another design uses a quantum feature map to calculate similarities for a classical classifier.
- The name QML describes a family of approaches. It does not promise better accuracy or speed.

The symbols carried between components

Symbol	What it is	This workshop's concrete value
x, y	Image and known label	784 normalized pixel values; digit 0 or 1
z	Classical representation	Four PCA components after a training-fitted transform
a	Encoding values	Four rotation angles in radians
θ	Model parameters	Sixteen trainable gate angles, separate from a
$U(a, \theta)$	Quantum circuit	Four qubits, two trainable layers, six logical CNOTs
n_0, n_1	Readout	Counts of 0 and 1 on measured qubit q_0
p, L	Prediction and error	$p = P(q_0 = 1)$ and binary cross-entropy loss

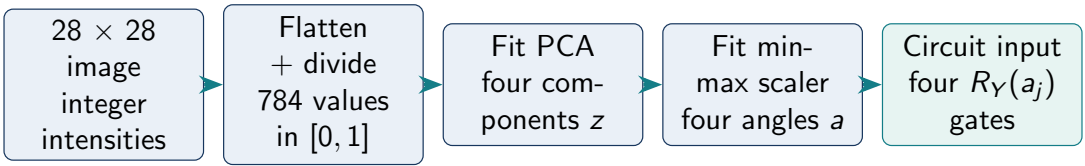
A feature is an input-dependent number. A weight is a learned model parameter; the two must never be confused.

Part 1: the dataset and its labels



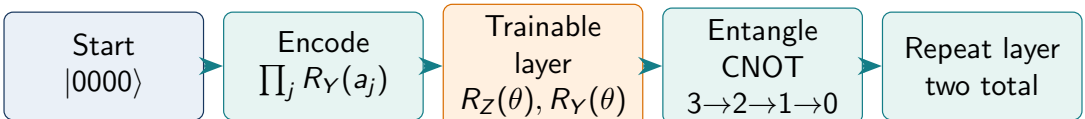
- The task here is handwritten 0 versus 1. Each example contains an image and a correct answer.
- The known label reaches the loss during training. An unseen image reaches the frozen forward path without its label.

Part 2: classical preprocessing



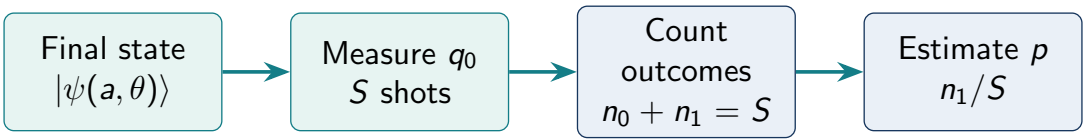
PCA and angle-scaling parameters are fit on training images only, then reused unchanged on validation, test, and IBM images.

Part 4: the trainable quantum circuit



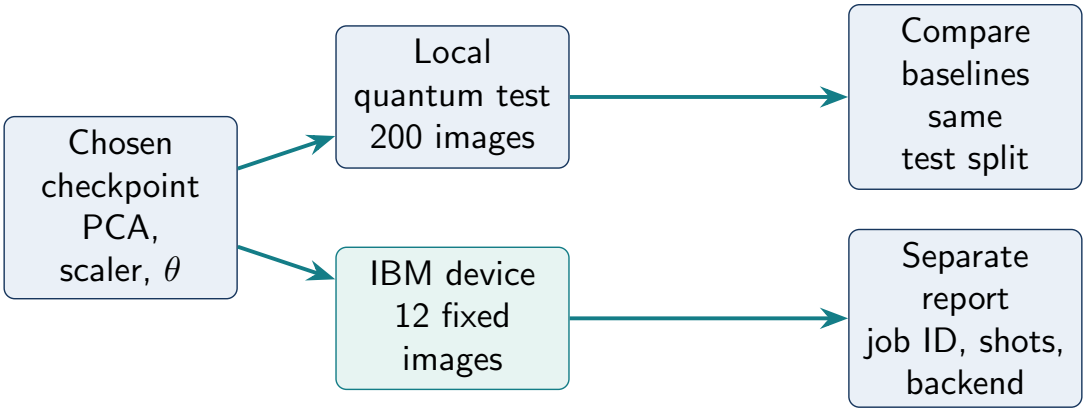
Each layer has eight learned rotation angles and three logical CNOT gates. Two layers make sixteen learned angles and six CNOTs before compilation.

Part 5: measurement becomes a prediction



Example: 768 ones in 1024 shots gives $p = 0.75$ and predicts digit 1. Those counts are illustrative, not IBM measurements.

Part 8: evaluation and the IBM device



Test accuracy estimates local generalization. The IBM run samples the frozen model on a prespecified 12-image subset; report it separately with backend, shots, job ID, and date.

Where each computation happens

Component	Runs on	Output
Load image, split, PCA, scale	Classical computer	Four angles a
Encode, trainable gates, CNOTs	Simulator or IBM device	Quantum state before measurement
Measure q_0	Simulator or IBM device	Bit counts or exact ideal probability
Probability, threshold, loss	Classical computer	Prediction and training objective
Choose new θ	Classical optimizer	New circuit parameters
Compare with classical baselines	Classical computer	Accuracy, loss, confusion matrix

Training is a repeated exchange between circuit evaluations and a classical optimizer. The main lab trains locally. The optional IBM section performs frozen inference and a small hardware update exercise.

The key difference is the prediction function

	Classical ML	Our QML model
Model parameters	Numerical weights	Gate angles
Forward pass	Arithmetic through layers	Prepare, rotate, entangle, measure
Prediction	Computed probability	Estimated measurement probability
Gradient	Chain rule through classical operations	Circuit derivatives via shifted executions, when used
Optimizer	Classical program	Classical program

Does QML use backpropagation?

- We must still learn how the loss changes when a parameter changes.
- A simulator can differentiate a circuit using several methods. Hardware returns measurement samples, not a complete computation graph of amplitudes.
- For a supported single rotation, the parameter-shift rule obtains a derivative from two shifted circuit evaluations.
- The chain rule then propagates this probability derivative through the loss and any classical components.
- Our main lesson uses COBYLA without gradients. Optional hardware training uses SPSA.

Two common approaches to QML

	Variational classifier	Quantum kernel
Quantum role	Trainable prediction circuit	Pairwise feature-state similarity
Classical role	Loss and optimizer	Kernel classifier such as SVM
Primary cost	Many image evaluations during training	Many pairwise kernel estimates
Today	Main implementation	Advanced explanation

Four papers that shaped the field

1. Biamonte et al., *Quantum machine learning*, Nature 549, 195–202 (2017). A broad survey of algorithms and possible applications.
2. Mitarai et al., *Quantum Circuit Learning*, Physical Review A 98, 032309 (2018). Trainable circuits with a hybrid optimization loop.
3. Schuld et al., *Circuit-centric quantum classifiers*, Physical Review A 101, 032308 (2020). A classifier built around parameterized circuits.
4. Schuld and Killoran, *Quantum machine learning in feature Hilbert spaces*, Physical Review Letters 122, 040504 (2019). Encoding as a feature map.

Full links and caveats are in the references at the end.

The papers connect to our five parts

Paper	What to notice in this workshop
Biamonte et al. (2017)	QML includes distinct algorithms and assumptions.
Mitarai et al. (2018)	Alternate quantum evaluations and classical parameter updates.
Schuld et al. (2020)	Use a trainable circuit to produce a classification readout.
Schuld and Killoran (2019)	Encoding data changes the effective feature space.
Havlíček et al. (2019)	Quantum feature maps can also feed a kernel classifier.

Multiple qubits use tensor products

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$$

- Two qubits have four basis amplitudes. Four qubits have sixteen.
- An n-qubit pure state has 2^n amplitudes, but a shot returns only n bits.
- The state-space dimension is not a free classical memory that can be read out completely.

Shots control sampling uncertainty

$$\widehat{p} = n_1/S, \qquad \mathrm{SE}(\widehat{p}) \approx \sqrt{p(1-p)/S} \leq \frac{1}{2\sqrt{S}}$$

- At 1024 shots, worst-case probability standard error is about 0.0156.
- Halving standard error requires approximately four times as many shots.
- More shots do not remove systematic hardware bias.
- Per-image probability uncertainty differs from uncertainty in test-set accuracy.

The exact data splits

Split	Total	Per digit	Source
Training	160	80	Official MNIST training set
Validation	40	20	Disjoint official training indices
Test	200	100	Official MNIST test set
Hardware demo	12	6	Fixed subset of the test split

The 12-image hardware subset is selected before seeing hardware predictions. It is not a full test benchmark.

The classical comparisons use the same split

- Main comparator: logistic regression on the same four scaled inputs.
- Additional reference: logistic regression on all 784 normalized pixels.
- The second comparison helps reveal the cost of compression.
- The models do not have matched parameter counts. State this limitation rather than implying an equal-capacity comparison.
- No accuracy claim is assumed before running the notebook.

PCA is learned from training data

- Center training vectors by their mean and identify orthogonal directions of maximum variance.
- Project each 784-pixel vector into the first four directions.
- Fit the PCA object on training images only. Use its transform method for validation and test.
- High variance does not necessarily mean high class-discriminative value.
- The notebook prints the explained variance rather than assuming four components retain everything.

Encoding choices have different costs

Encoding	Representation	Important cost
Basis	Bits in basis states	Qubits and possible discretization
Angle	Values as gate rotations	Qubits or repeated data loading
Amplitude	Normalized values as amplitudes	Generic state preparation can be expensive

We use angle encoding because its circuit and input scaling are easy to inspect.

The four-qubit classifier

$$|\psi(a,\theta)\rangle = U(\theta)U_\phi(a)|0000\rangle$$

- Initialize four qubits to zero.
- Encode feature a_j with $R_y(a_j)$ on wire j.
- Apply two trainable layers, each containing RZ followed by RY on every wire.
- After each layer, use CNOTs 3 to 2, 2 to 1, and 1 to 0.
- Measure wire 0 to obtain the class-1 probability.

A numerical measurement-to-loss example

$$n_1 = 768, \quad S = 1024, \quad \widehat{p} = 0.75, \quad \widehat{\langle Z \rangle} = 1 - 2(0.75) = -0.5$$

- The fixed 0.5 threshold predicts digit 1.
- If the true label is 1, its loss is $-\log(0.75) \approx 0.288$.
- If the true label is 0, its loss is $-\log(0.25) \approx 1.386$.
- This is a worked illustrative count example, not a reported hardware result.

The hybrid training loop

- For each training image, bind the feature angles and current parameters.
- Compute ideal probabilities in the local training route.
- Average binary cross-entropy across training images.
- Ask a classical optimizer for a new parameter vector.
- Every fifth objective evaluation, evaluate validation loss and retain the best checkpoint.

Parameter-shift gives exact circuit derivatives

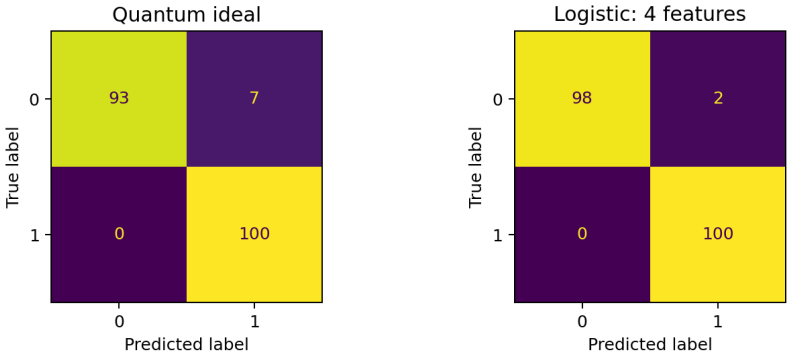
$$\frac{\partial p}{\partial \theta_j} = \frac{p(\theta_j + \pi/2) - p(\theta_j - \pi/2)}{2}$$

- This form applies when a parameter appears once in a Pauli-generated rotation.
- Each of our trainable angles appears once, so the rule applies.
- The notebook checks one derivative against a finite-difference estimate.
- Shared parameters or other generators require the corresponding generalized derivative rule.

The model is frozen before the final test

- Report local ideal performance on all 200 selected test images.
- Compare the same four-feature inputs against logistic regression.
- Inspect confusion matrices and probabilities, not only one accuracy number.
- Keep the 12-image hardware demonstration separate from the 200-image test.
- No result from this small demo establishes general MNIST superiority.

Actual held-out confusion matrices



Both models use the same four input features and the same held-out examples.

Local installation and notebook launch

```
python -m venv qml-env
# macOS/Linux:
source qml-env/bin/activate
# Windows PowerShell: qml-env\Scripts\Activate.ps1
python -m pip install -r requirements.txt
python -m pip install jupyterlab
python -m jupyter lab
```

In Colab, upload the notebook and the two data/checkpoint files. Run its installation cell once and restart if needed.

Notebook laboratory: explicit parameter binding

```
mapping = {p: float(v)
           for p, v in zip(features, angles)}
mapping.update({p: float(v)
                for p, v in zip(weights, theta)})
bound = circuit.assign_parameters(mapping)
readout = ClassicalRegister(1, 'readout')
bound.add_register(readout)
bound.measure(0, readout[0])
```

Binding by object avoids accidentally swapping feature and weight positions.

Exact prediction on a classical simulator

```
state = Statevector.from_instruction(
    bind_circuit(angles, theta))
p_one = state.probabilities()[1::2].sum()
```

Odd statevector indices have the least significant qubit $q_0 = 1$.

This calculation uses the unmeasured circuit. Real hardware instead estimates the probability from repeated measurements.

What a QNode does

- A QNode wraps a quantum function so PennyLane can run it on a chosen device.
- `default.qubit` is a local simulator. It is not an IBM quantum computer.
- The QNode specifies quantum operations and a measurement.
- An automatic differentiation interface can connect circuit outputs to a classical loss.

Three execution modes answer different questions

Mode	What it includes	What it omits
Exact statevector	Ideal circuit probabilities	Shot noise and device errors
Finite-shot simulator	Sampling variability	Device errors unless modeled
Physical hardware	Actual device and sampling effects	Direct access to the full quantum state

Account preparation belongs before the live session

- Sign in to `https://quantum.cloud.ibm.com/`.
- Confirm an accessible compute instance, region, and current quota.
- Create an API key and enter it through a hidden prompt or an environment variable.
- Optionally specify the instance CRN to select the intended instance.
- Never paste an API key into slides, shared notebooks, or screenshots.

Connect with the current Runtime service

```
token = getpass.getpass('IBM Quantum API key: ')
service = QiskitRuntimeService(
    channel='ibm_quantum_platform',
    token=token)
backend = service.least_busy(
    operational=True, simulator=False,
    min_num_qubits=4)
```

The notebook also supports an instance CRN and an explicit backend name. API documentation checked 12 November 2026.

Compile the measured, bound circuits

```
pm = generate_preset_pass_manager(
    backend=backend, optimization_level=1,
    seed_transpiler=17)
isa_circuits = pm.run(measured)
assert all(c.num_parameters == 0
           for c in isa_circuits)
```

A physical circuit object may have many wires after layout. Our logical algorithm still uses four qubits.

Software sources and critical perspective

- PennyLane QNode and parameter-shift documentation:
  `https://docs.pennylane.ai/en/stable/code/api/pennylane.QNode.html`
- IBM Quantum Learning, *Quantum machine learning*:
  `https://quantum.cloud.ibm.com/learning/en/courses/quantum-machine-learning`
- IBM SamplerV2:
  `https://quantum.cloud.ibm.com/docs/en/api/qiskit-ibm-runtime/sampler-v2`
- Cerezo et al., *Challenges and opportunities in quantum machine learning*, Nature Computational Science 2, 567–576 (2022).
  `https://doi.org/10.1038/s43588-022-00311-3`

Read a result in the context of its data access, noise model, baseline, and resource assumptions. A workshop run is a demonstration rather than a claim of quantum advantage.

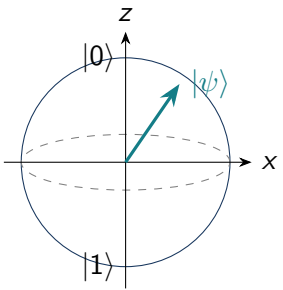
Vectors and the Born rule

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad |\psi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

- Example: $\alpha = \sqrt{3}/2$ and $\beta = 1/2$.
- Predicted outcome probabilities are 0.75 and 0.25.
- Finite samples fluctuate around those probabilities.

The Bloch sphere organizes single-qubit states

$$|\psi\rangle = \cos(\vartheta/2)|0\rangle + e^{i\varphi}\sin(\vartheta/2)|1\rangle$$



- Polar angle controls computational-basis probabilities.
- Relative phase changes direction around the sphere.
- A global phase does not change physical predictions.
- Pure states lie on the surface. Mixed states can lie inside.

X flips a computational-basis state

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \qquad X|0\rangle = |1\rangle$$

- X is the quantum counterpart of a bit flip on basis states.
- It also transforms superpositions by swapping their amplitudes.
- Applying X twice returns the original state.

RY turns a number into a measurement probability

$$R_y(a) = \begin{bmatrix} \cos(a/2) & -\sin(a/2) \\ \sin(a/2) & \cos(a/2) \end{bmatrix}$$

$$R_y(a)|0\rangle = \cos(a/2)|0\rangle + \sin(a/2)|1\rangle, \qquad P(1) = \sin^2(a/2)$$

- Angles 0, $\pi/2$, and π give probabilities 0, 0.5, and 1.
- This makes rotations a direct way to encode numeric features.

A Bell state shows entanglement

$$|00\rangle \xrightarrow{H\otimes I} \frac{|00\rangle + |10\rangle}{\sqrt{2}} \xrightarrow{\text{CNOT}} \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

- The Bell state cannot be written as a product of two individual pure-qubit states.
- Computational-basis shots produce correlated 00 and 11 outcomes.
- Correlations in one basis alone do not prove entanglement experimentally.
- Entanglement is a possible modeling resource, not a guarantee of classification advantage.

Qiskit ordering needs explicit attention

- Qiskit prints multi-qubit basis strings as $q_{n-1} \dots q_1 q_0$.
- For four qubits, odd statevector indices correspond to $q_0 = 1$.
- Our classifier measures only q_0 into a one-bit register named `readout`.
- After transpilation, the logical-to-physical qubit mapping can change, while the classical output mapping remains defined.

PCA as a concrete matrix operation

$$z = (\widetilde{x} - \mu)W^T, \qquad W \in \mathbb{R}^{4\times 784}, \quad z \in \mathbb{R}^4$$

- μ is the mean training image vector.
- The four rows of W are the retained PCA directions.
- The same μ and W transform every future image.
- Saving only the quantum angles is insufficient for later predictions from raw pixels.

Why we do not encode 784 pixels directly

- One angle per raw pixel would require 784 rotations and either many qubits or repeated encoding blocks.
- Amplitude encoding can fit 784 padded values into ten qubits, but state preparation is not generally free.
- Four-component PCA offers a transparent teaching compromise.
- The compression is classical and discards information. We quantify its effect with baselines.

A checkpoint needs more than circuit weights

- Save the trained angles and circuit configuration.
- Save the PCA mean, PCA components, and scaler coefficients.
- Record split indices, dataset hash, random seed, package versions, and validation selection rule.
- For hardware, also retain backend name, job ID, shot count, and the matching model checkpoint.
- Credentials never belong in saved experiment artifacts.

Hardware training has a substantial measurement budget

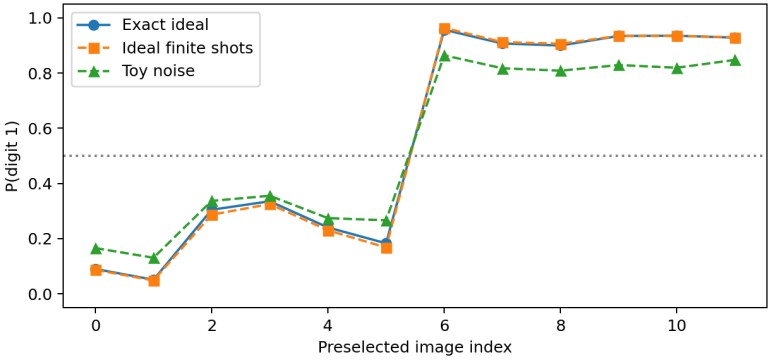
$$2 \times 16 \times 160 = 5120 \text{ shifted image circuits per full gradient}$$

- At 1024 shots each, those shifted evaluations use 5,242,880 circuit shots.
- Unshifted loss evaluations add more work.
- Queue overhead, transpilation, device execution, and account policy all matter.
- This motivates local pretraining and a deliberately small optional hardware update exercise.

Noise sources affect different parts of the circuit

- Relaxation moves excited-state population toward the ground state.
- Dephasing reduces phase coherence.
- Imperfect gates alter the intended transformation.
- Readout mistakes can flip the reported classical bit.
- Calibration drift means device behavior can change between runs.

A controlled toy-noise experiment



Illustrative error model only: 0.2% one-qubit depolarization, 2% two-qubit depolarization, and asymmetric 3%/4% readout errors. These are not IBM calibration measurements.

Confidence intervals help identify unstable predictions

$$\text{Wilson center} = \frac{\widehat{p} + z^2/(2S)}{1 + z^2/S}, \qquad z = 1.96$$

- The notebook computes the full Wilson interval for each measured probability.
- An interval crossing 0.5 indicates uncertainty relative to the decision threshold.
- The calculation assumes independent Bernoulli shots.
- Drift and correlated errors are not captured by a simple binomial interval.

Optional exercise: parameters updated from hardware

- Start from the simulator-trained checkpoint.
- Use four predetermined training images, two from each class.
- Perform two SPSA steps using 512 shots per circuit.
- Actual hardware counts determine the loss difference and parameter update.
- This is a fine-tuning demonstration, not a complete hardware training experiment.

SPSA uses two perturbed loss evaluations

$$\Delta_j \in \{-1, +1\}, \quad L_{\pm} = L(\theta \pm c\Delta)$$

$$\widehat{g}_j = \frac{L_+ - L_-}{2c\Delta_j}, \qquad \theta_j \leftarrow \theta_j - a\widehat{g}_j$$

- Two evaluations perturb all parameter directions simultaneously.
- The estimate is noisy. Repeated steps and tuned schedules are needed in serious training.
- Our teaching defaults are $a = 0.05$ and $c = 0.15$.

Ten-class MNIST changes the output model

- Include all ten digits with stratified train/validation/test splits.
- Refit PCA and scaling on the new training set.
- Use a ten-class readout and categorical cross-entropy.
- Measure a full confusion matrix and per-class metrics.
- Binary-to-multiclass conversion requires retraining and more than changing a label filter.

Quantum kernels compare encoded feature states

$$K(x, z) = |\langle \phi(z) | \phi(x) \rangle|^2$$

- Prepare $U_\phi(x)|0\rangle$, apply $U_\phi(z)^\dagger$, and measure the all-zero probability.
- A classical SVM can use the estimated Gram matrix.
- Pairwise kernel estimation can require quadratic work in the training-set size.
- Compare with classical kernels and account for finite-shot matrix errors.

Barren plateaus and trainability

- In some circuit families, gradient variance becomes very small as the system grows.
- Depth, initialization, cost locality, and noise can affect trainability.
- A four-qubit success does not show that the same architecture scales.
- Use physically informed shallow circuits and examine gradient/optimization behavior.
- Increasing the number of qubits alone is not a research contribution.

Common software and data failures

Symptom	Action
Import/version conflicts	Create a fresh environment and install the tested requirements.
MNIST download blocked	Use the bundled real-MNIST subset.
Checkpoint mismatch	Restore matching preprocessing or retrain.
Missing notebook variable	Restart and run cells in order.
Unexpected parameter order	Bind with parameter objects in a dictionary.

Common IBM execution failures

Symptom	Action
Authentication fails	Verify API key, account, region, and instance access.
No available backend	Check accessible devices and quota. Continue locally.
Circuit rejected	Transpile against the selected backend and bind all parameters.
Missing result field	Use the actual named register, `readout`.
Queue is long	Save the job ID and retrieve later without duplicating submission.

Thank You!

Questions & Discussion

Mohammad Saidur Rahman, Ph.D.

Jesus Lopez Viviana Cadena

Intelligent and Quantum Secure Advanced Cyber Defense Research (IQSeC) Lab
University of Texas at El Paso

rahmanmsaidur.com