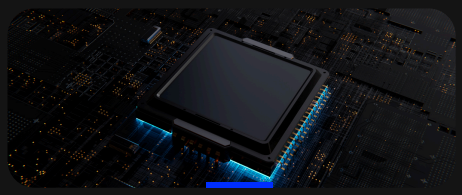
Quantum Time Series Forecasting with Qiskit

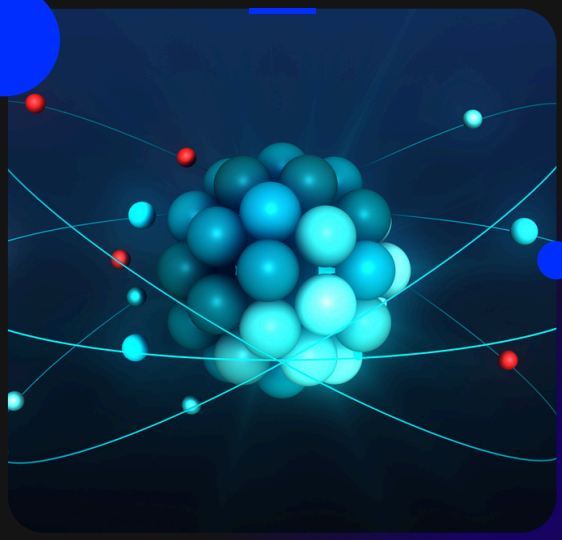
Name : Harshita Deewan
Presented Under The Guidance Of : Dr. Trivedi Harsh Chandrakant
College : LNMIIT, Jaipur



Hello everyone! Welcome to this session on Quantum Time Series Forecasting with Qiskit. Over the next hour, we'll explore how quantum computing – a field once thought purely theoretical – is now beginning to intersect with data science problems like time series forecasting.
I'm Harshita Deewan, and I've been learning about quantum computing applications in machine learning.
Today's talk is a blend of quantum theory, machine learning, and code – all using IBM's open-source toolkit, Qiskit.

What is Time Series Forecasting?

- Predict future values using past data
- Examples: Stock prices, temperature, energy load
- Applications: finance, healthcare, IoT



Time series forecasting is about modeling data that evolves over time.
Think of predicting stock prices, weather changes, or electricity demand.
These forecasts drive important decisions – like when to trade, when to conserve energy, or even when to trigger health alerts.
But the challenge is: how can we make these forecasts more accurate, especially when the data is noisy or shows complex patterns?

A time series is made up of different elements:

- Trend refers to a consistent upward or downward direction.

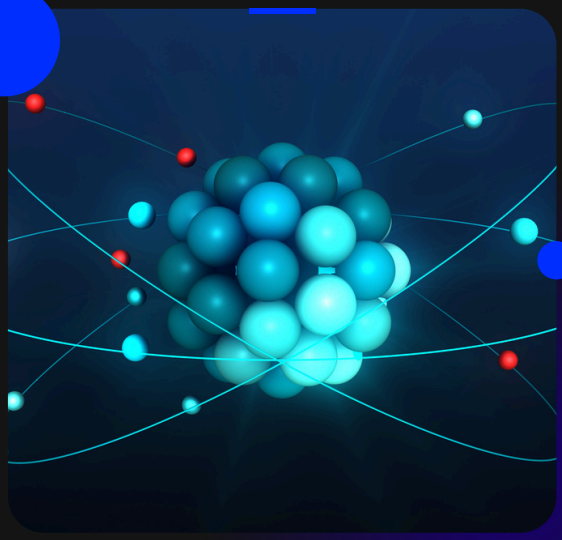
- Seasonality shows up as repeating patterns – daily, weekly, or yearly.

- Noise is random variation – the hardest to model.

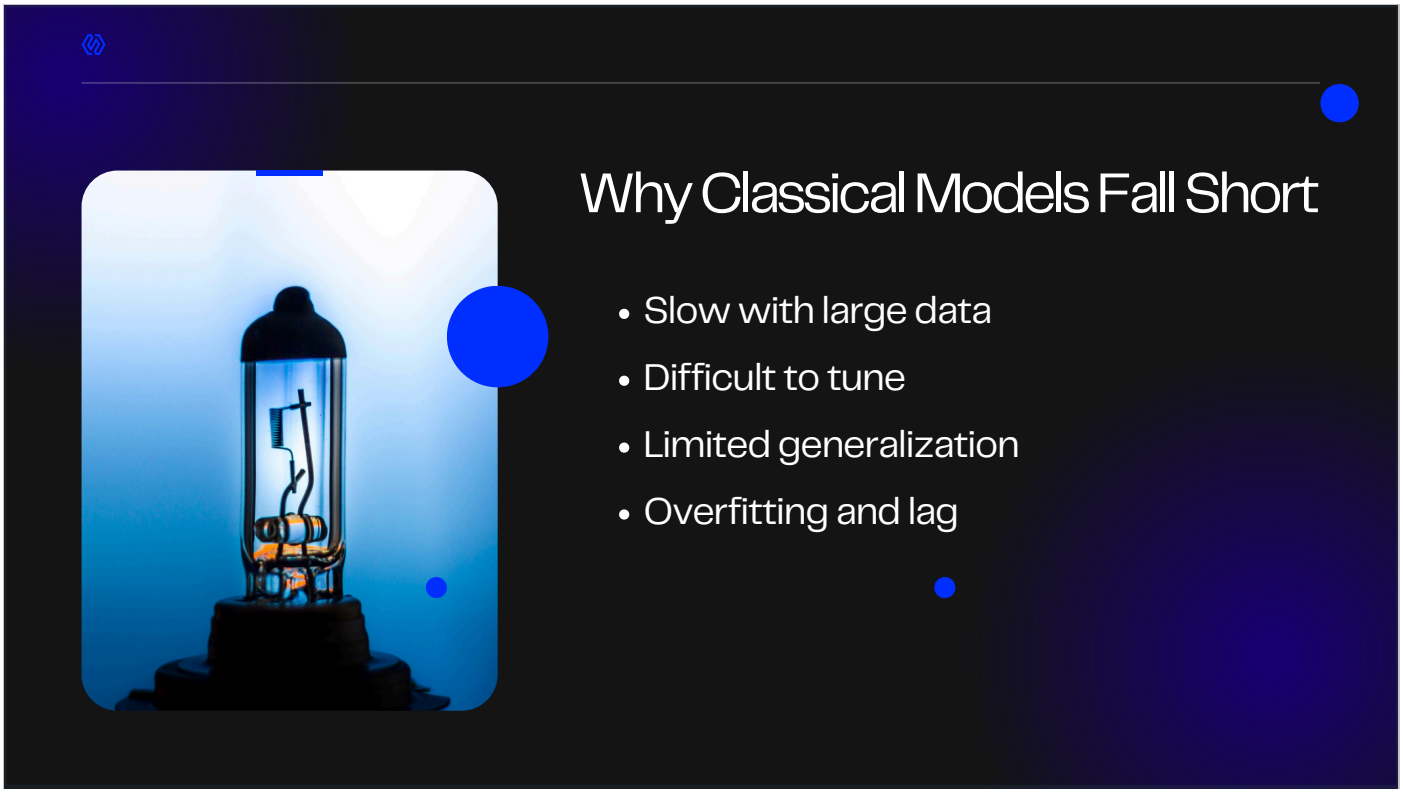
- And stationarity is often required for certain models to work well, meaning the mean and variance stay consistent over time.
In real-world data, all of these exist together – and that's where it gets tricky.

Classical Forecasting Models

- ARIMA, Holt-Winters, Exponential Smoothing
- LSTM, Prophet
- Strengths and when to use



Traditionally, forecasting is handled using statistical models like ARIMA or Holt-Winters.
Then came deep learning models like LSTM that capture long-term dependencies in sequences.
Tools like Prophet, by Facebook, automate trend and seasonality detection.
However, all of these have computational and modeling limits – especially with non-stationary, complex datasets.

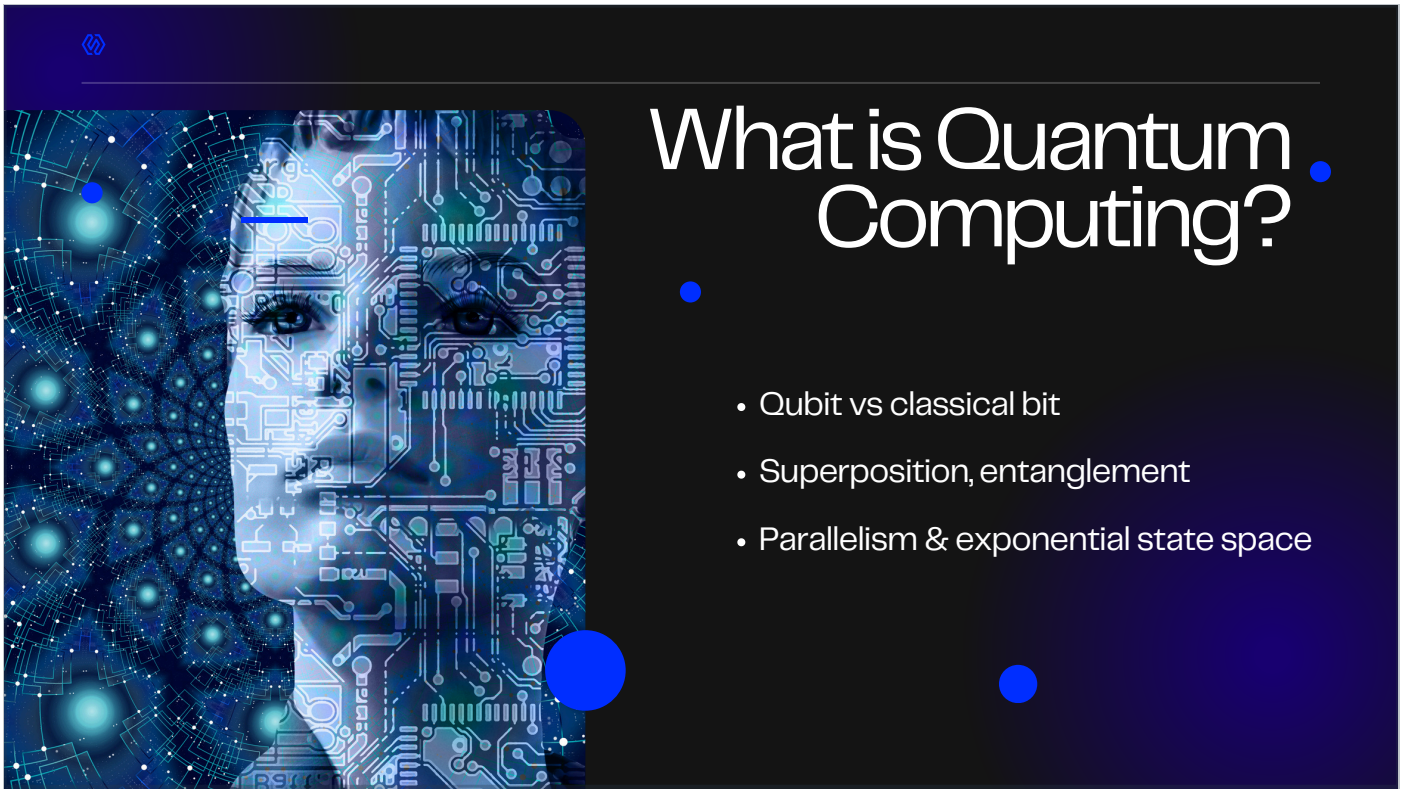


These models are powerful, but they’re not perfect:

- They scale poorly with big, high-dimensional datasets.

- LSTMs need a lot of data and careful tuning.

- They’re also prone to overfitting, especially when data is sparse or non-stationary. Quantum machine learning aims to address some of these challenges with a new computational paradigm.

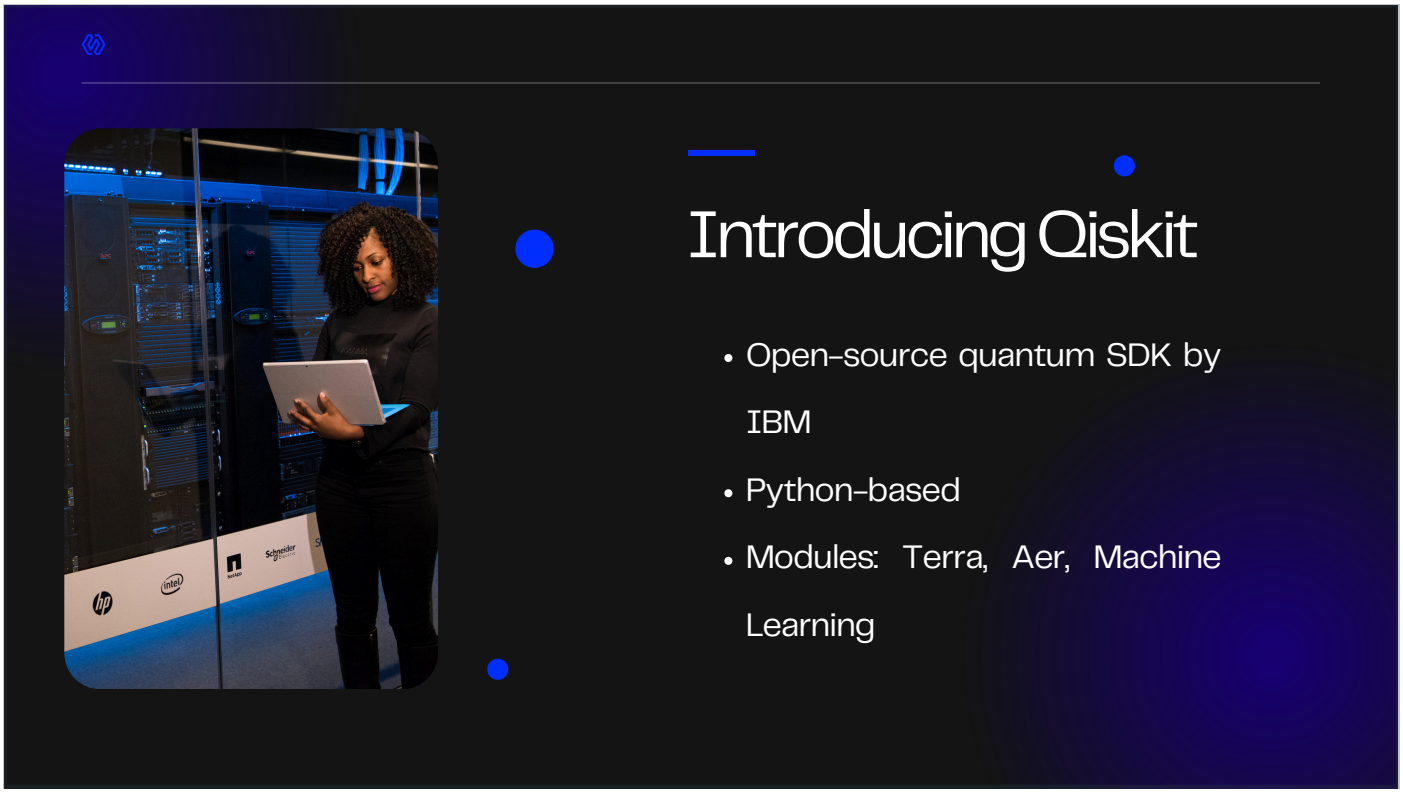


Unlike a classical bit, a qubit can exist in multiple states at once due to superposition.
Two qubits can become entangled, meaning their states are linked even across distance.
Quantum interference helps amplify correct paths in computation.
This leads to massive parallelism – and that's where machine learning benefits.

Why Quantum for ML?

- High-dimensional computation
- Speedups for learning
- Quantum-native probabilistic modeling

Machine learning often needs to handle high-dimensional vectors, large matrices, and probability distributions.
Quantum computers can encode and manipulate these more naturally and sometimes more efficiently.
For time series, this means we might be able to learn more from less data, or find patterns classical models miss – especially in noisy or uncertain environments.



IBM’s Qiskit is a leading open-source SDK for building quantum algorithms.
It’s Python-based and easy to use for developers and researchers.
Key modules include:

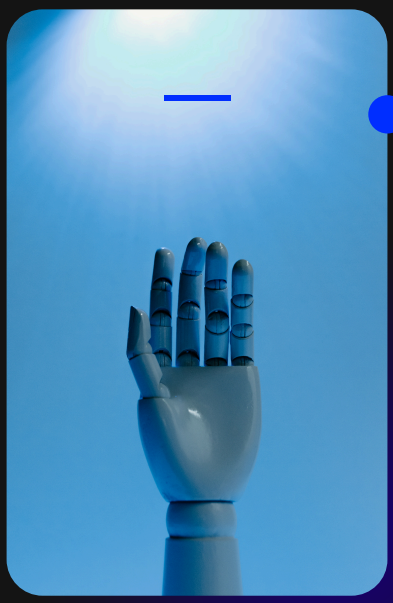
- Terra for circuits,

- Aer for simulation,

- and Qiskit Machine Learning, which we’ll focus on today.

Qiskit Machine Learning Tools

- Quantum classifiers, regressors
- Variational circuits
- Hybrid ML workflows



With Qiskit ML, we can define quantum neural networks, plug them into scikit-learn pipelines, and build hybrid models.
These tools make it possible to train quantum models on classical data – such as time series – and forecast future points using quantum circuits.

The general workflow is:

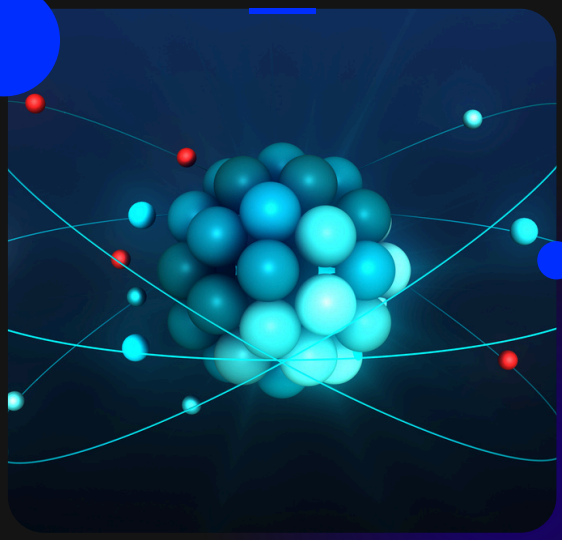
- Encode your time series data into quantum circuits.

- Use a model like a Variational Quantum Circuit (VQC) or Quantum Kernel Regressor.

- Forecast the next steps by measuring outputs and comparing predictions with actual values.

Encoding Time Series into Qubits

- Amplitude encoding: normalized vectors
- Angle encoding: rotation gates (Ry, Rz)
- Basis encoding: discrete states

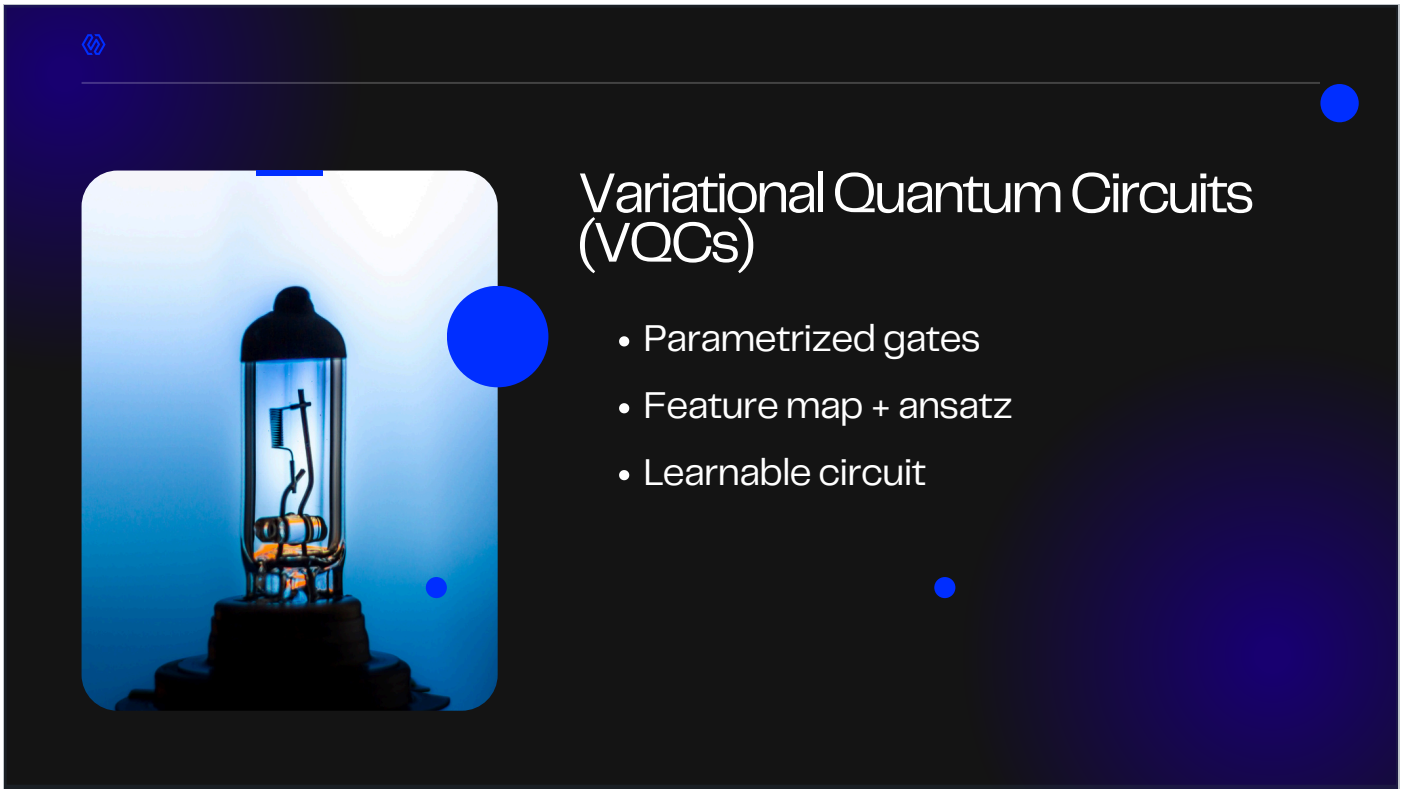


We need to translate our classical data into quantum states:

- Amplitude encoding puts the entire dataset into amplitudes – very efficient but hard to implement.

- Angle encoding is simpler and uses qubit rotations to reflect data.

- Each method has trade-offs: complexity, noise tolerance, and scalability.



VQCs are the backbone of quantum ML.
They consist of:

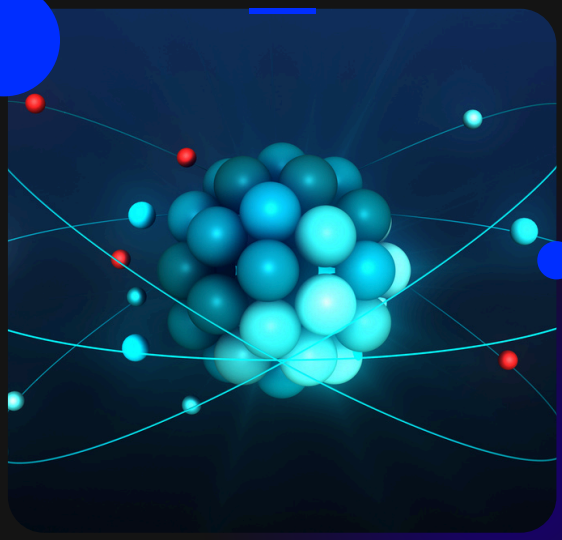
- A feature map to encode the input,

- An ansatz – a learnable quantum circuit,

- And measurement operators.
These circuits are trained via a classical optimizer using gradient descent.

Qiskit Regressor Architecture

- EstimatorQNN + NeuralNetworkRegressor
- Hybrid training loop
- Compatible with scikit-learn



Qiskit's NeuralNetworkRegressor wraps around a quantum neural network defined using EstimatorQNN.
We get compatibility with all scikit-learn features: cross-validation, pipelines, scoring.
This makes it easier to integrate quantum models into your data science workflows.

Code Setup – Imports & Dataset

```
import numpy as np
from sklearn.model_selection import train_test_split
from qiskit.circuit.library import ZZFeatureMap, RealAmplitudes
from qiskit_machine_learning.neural_networks import EstimatorQNN
from qiskit_machine_learning.algorithms import NeuralNetworkRegressor
```

- Load dependencies
- Use EstimatorQNN for quantum regression
- Use synthetic sine wave data

Prepare Synthetic Time Series

Step 1: Create Sample Time Series

```
x = np.linspace(0, 2 * np.pi, 50)
y = np.sin(x) + 0.1 * np.random.randn(50)

X = np.column_stack((x[:-1], x[1:]))
y_target = y[1:]

X_train, X_test, y_train, y_test = train_test_split(X, y_target, test_size=0.2)
```

- Generate noisy sine wave
- Create feature pairs (sliding window)
- Train–test split

Define Quantum Circuit

Step 2 : Define Quantum Neural Network

```
feature_map = ZZFeatureMap(feature_dimension=2, reps=2)
ansatz = RealAmplitudes(num_qubits=2, reps=2)

qnn = EstimatorQNN(
    circuit=feature_map.compose(ansatz),
    input_params=feature_map.parameters,
    weight_params=ansatz.parameters
)
```

- ZZFeatureMap : encodes time steps
- RealAmplitudes : learnable circuit
- Compose both to form the QNN

Train Quantum Regressor

Step 3: Train Quantum Regressor

```
regressor = NeuralNetworkRegressor(qnn=qnn)
regressor.fit(X_train, y_train)
```

- Wrap QNN in a NeuralNetworkRegressor
- Follows scikit-learn conventions
- Uses classical optimization (COBYLA/SPSA)

Predict & Visualize

Step 4: Predict and Plot Results

```
import matplotlib.pyplot as plt

y_pred = regressor.predict(X_test)

plt.scatter(range(len(y_test)), y_test, label="True")
plt.scatter(range(len(y_pred)), y_pred, label="Predicted")
plt.legend()
plt.title("Quantum Forecasting Result")
plt.show()
```

- Compare forecast vs. actual
- Evaluate visually due to small sample size
- RMSE/MAE can be shown optionally

Evaluate Performance

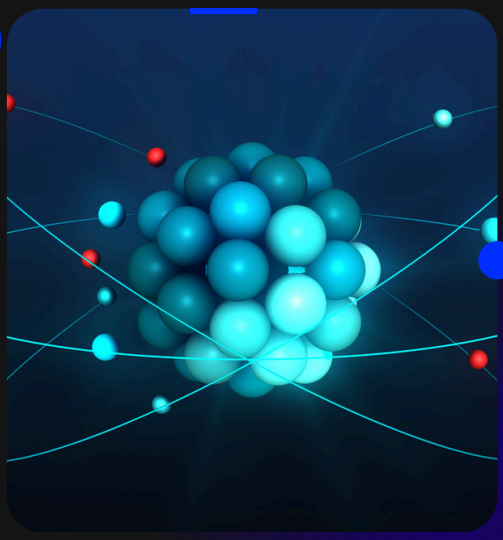
Evaluation : MAE and RMSE

```python
from sklearn.metrics import mean_absolute_error, mean_squared_error

mae = mean_absolute_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)

print(f"MAE: {mae:.3f}, RMSE: {rmse:.3f}")
```

Case Study: Quantum Forecasting Results

How did the quantum regressor perform?

- Dataset: Noisy sine wave, 50 samples
- Model: Variational Quantum Regressor (VQR) with 2 qubits
- Compared with:
- Classical Linear Regression
- LSTM (on same data)
- Evaluation metrics: MAE & RMSE



Here we see a direct comparison:

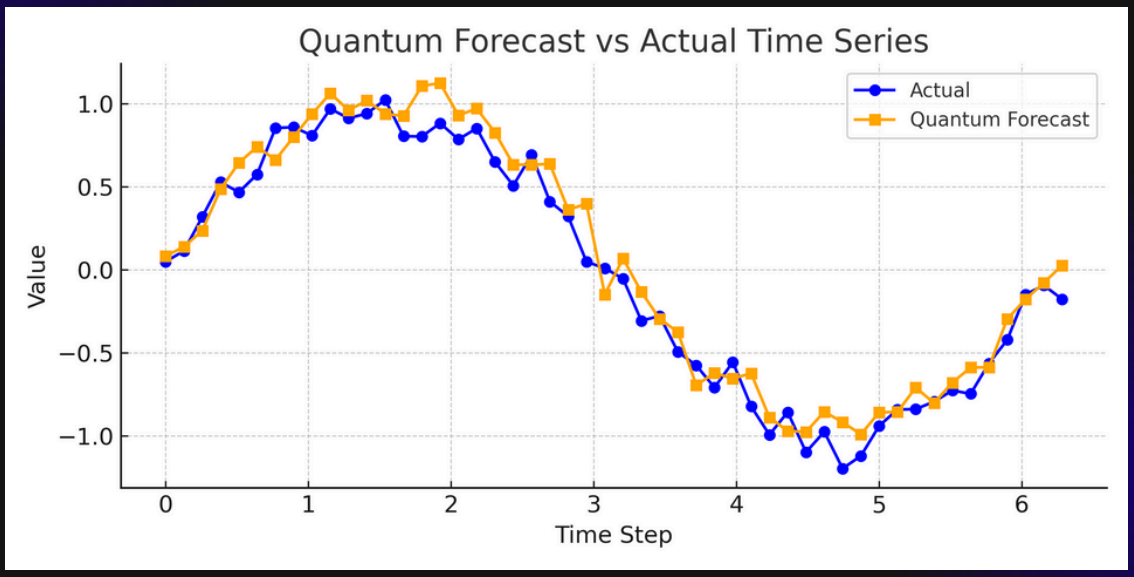
- Despite having only 2 qubits and a small dataset, the quantum model predicted reasonably well.

- RMSE and MAE were close to classical Linear Regression, slightly higher than LSTM which had more capacity but also risked overfitting.

- This shows the promise of quantum models, especially for small datasets or when classical models struggle with high-dimensional features.

Visualization : Forecast vs Actual

Visual Comparison



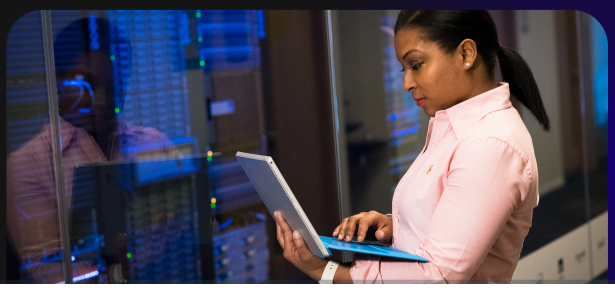
Visually, the quantum model captures the trend quite well, even if it sometimes under- or overshoots.
The gaps remind us that this is a shallow circuit (2 qubits, 2 reps).
With deeper circuits and more qubits, accuracy could improve — but today's hardware limits us.

Quantum circuits can embed complex, nonlinear relationships that might need hundreds of classical neurons.
They also have the potential to generalize better with fewer data points, as information is distributed over entangled qubits.
While practical speedups are future-facing, the structural benefits are real even now.

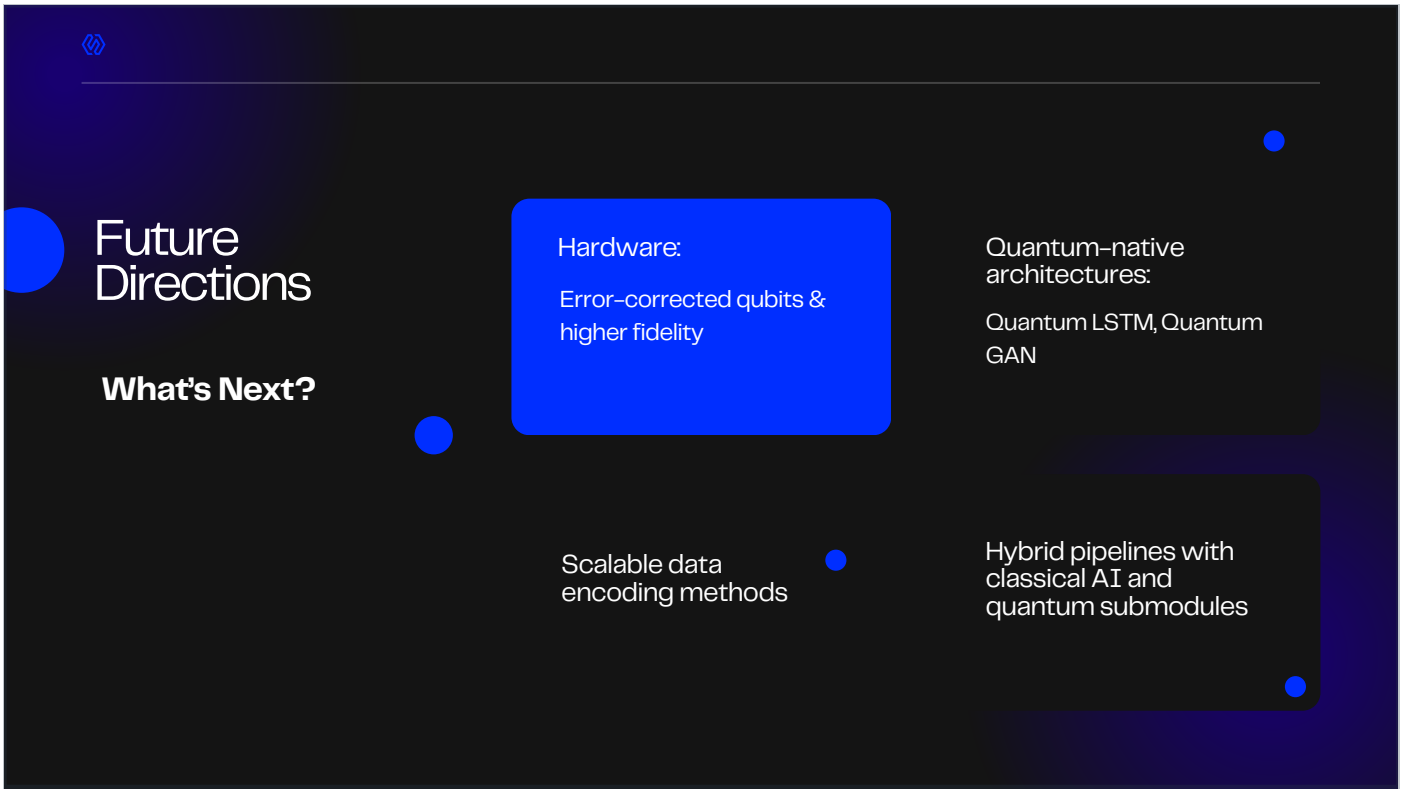
Challenges and Limitations

Current Barriers

- Noisy Intermediate-Scale Quantum (NISQ) devices
- Limited number of qubits (≈ 5–100)
- Circuit depth limited by noise
- Data encoding overhead
- Hybrid models still rely heavily on classical compute



Our models are limited by hardware: noise, shallow depth, and qubit count.
Encoding classical data into quantum states can also consume a lot of resources.
And although training is hybrid, the classical part can still be the bottleneck.
But researchers are actively improving both algorithms and hardware.



The field is moving fast:

- IBM, Google, and startups are building better qubits.

- Researchers are exploring quantum-native architectures like quantum LSTMs, which directly model sequences.

- Hybrid models might handle large data with classical preprocessing + quantum forecasting modules.

Thank You

The only limit is the imagination.