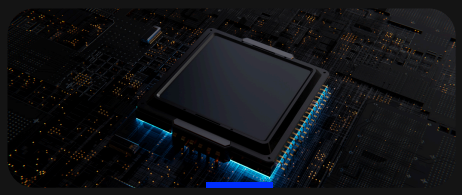
Quantum Time Series Forecasting with Qiskit

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



Hello everyone, and thank you for joining me today. I'm Harshita Deewan from LNMIIT, Jaipur, and I'm delighted to present this talk titled 'Quantum Time Series Forecasting with Qiskit', guided by Dr. Trivedi Harsh Chandrakant.

The world is currently undergoing a technological transformation, and quantum computing is at the forefront. While it was once an abstract theoretical concept, today it's becoming a practical tool for real-world applications.

At the same time, machine learning — and especially time series forecasting — has become a fundamental part of how businesses make predictions, optimize performance, and prepare for the future. From financial markets to climate modeling and smart grids, time series forecasting is every where.

What if we could combine the strengths of quantum computing with the predictive power of machine learning? That's the essence of today's discussion.

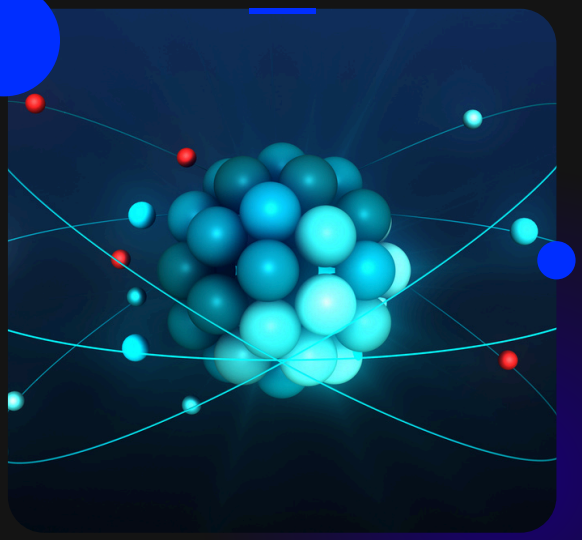
We'll begin with an introduction to time series forecasting and classical modeling techniques. Then, I'll walk you through the limitations of these classical approaches and show how quantum computing offers potential improvements. Next, we'll explore how to use Qiskit, IBM's open-source SDK, to build a quantum regressor. We'll go through real code, run a quantum forecasting experiment, and analyze its results.

Whether you're new to quantum computing or looking to apply it to data science, I hope this talk

provides both clarity and inspiration. Let’s dive in.

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 the process of predicting future values based on previously observed data points that are ordered in time. It's a specialized form of supervised learning where the input features are sequences and the outputs are the future values of that same sequence.

A time series could be stock prices changing daily, electricity demand fluctuating hourly, or temperature rising and falling seasonally.

The goal here isn't just to model a single value, but to understand the pattern behind the changes — the dependencies, the lags, and the seasonality.

This forecasting capability is used across domains:

In finance, time series helps predict prices, detect anomalies, and assess risk.

In healthcare, we use it for predicting vital signs, hospital admissions, or disease progression.

In IoT and smart cities, forecasting helps manage energy consumption and anticipate load.

Time series data is special because it has a temporal dependency. The order matters. Unlike image data or static tables, past values influence future values — and that makes the modeling task both powerful and challenging.

Throughout this presentation, we'll examine how both classical and quantum methods can handle

these temporal patterns, and what advantages quantum circuits might offer in representing such dynamic systems.

Understanding the anatomy of a time series is essential before we can forecast it accurately. A time series typically has four key components:

1. Trend: This refers to the long-term movement in the data. It could be increasing, decreasing, or stable. For example, inflation causes a gradual upward trend in prices over years.

2. Seasonality: These are regular, repeating patterns. Think of temperature changes over seasons, or increased online shopping during festivals. Seasonality helps in predicting values based on cyclical behaviors.

3. Noise: This is the random, unexplained part of the data. It's often caused by external, unpredictable events — like a market crash or sudden weather change. Noise makes forecasting more difficult and can lead models to overfit.

4. Stationarity: A stationary time series has constant mean, variance, and autocorrelation over time. Many forecasting models assume stationarity because it makes the mathematics simpler. However, real-world data is rarely stationary, which requires pre-processing like differencing or tra nsformation.

When we model a time series, we're often trying to separate signal from noise, trend from randomness. Accurately capturing these components allows us to build better forecasts.

Later, we'll see how quantum models can help capture non-linear patterns across time steps more

compactly, even with minimal data.

Over the years, several classical models have been developed to tackle time series forecasting. Let's take a look at some key types:

ARIMA (AutoRegressive Integrated Moving Average): This statistical model is based on the idea that past values and past prediction errors can be linearly combined to forecast the next step. It works well for stationary data, but requires careful preprocessing and parameter tuning (p, d, q).

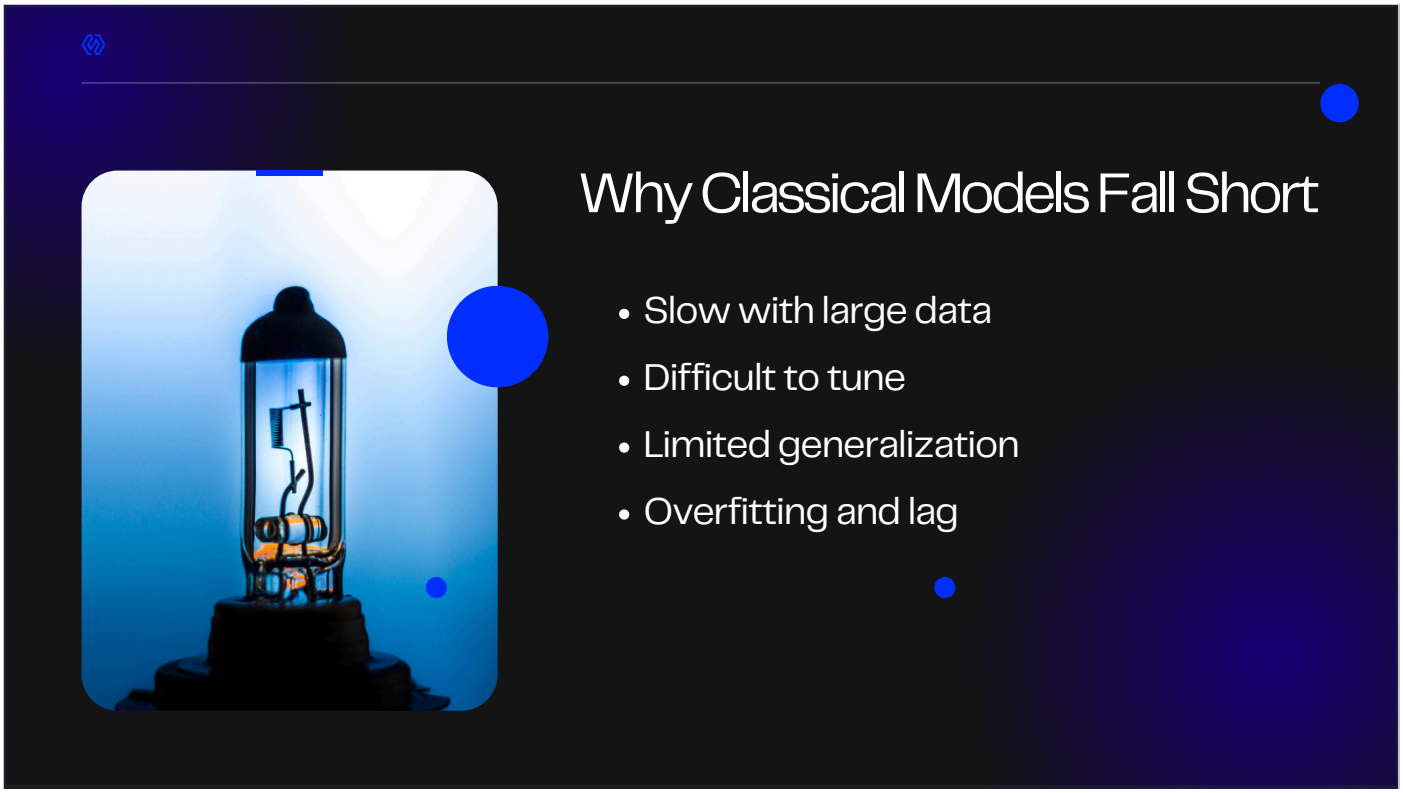
Holt-Winters Exponential Smoothing: This method is great for data with seasonality and trend. It applies exponential weighting to past observations, giving more importance to recent data.

LSTM (Long Short-Term Memory): A type of recurrent neural network, LSTMs are powerful because they can learn long-term dependencies. They use memory cells and gates to store and update information across time steps. However, LSTMs require a lot of data and compute power.

Prophet: Developed by Facebook, Prophet is known for being user-friendly. It automatically detects trend and seasonality, and is used widely in business forecasting.

These models each have strengths and weaknesses. Simpler models are easier to interpret and faster to train but may not capture complex dependencies. Deep learning models can be powerful but are resource-intensive and hard to interpret.

Later, we'll compare these with quantum models, which offer a unique blend of compact representation and generalization potential.



While classical forecasting models like ARIMA, LSTM, and Prophet have proven effective in many scenarios, they come with a number of challenges and limitations, especially as datasets become more complex, noisy, or high-dimensional.

Let's go through some of the key issues:

Scalability: Classical models can struggle when we scale up either the data volume or the number of features. ARIMA, for example, works well for univariate time series but gets computationally expensive and unstable for multivariate series. LSTMs handle multivariate data but require significant memory and computational resources to train. When dealing with long historical records or streaming data, training and inference times can grow dramatically.

Data Requirements and Overfitting: Deep learning models, especially LSTMs, are data-hungry. They need large datasets to generalize well. But in many real-world cases — like medical forecasting, rare financial events, or anomaly detection — we simply don't have massive datasets. In such cases, classical models can easily overfit, especially when the signal-to-noise ratio is low.

Hyperparameter Tuning: Classical models often come with many hyperparameters that need to be set just right. ARIMA needs p, d, q; LSTMs need tuning of hidden units, learning rates, dropout rates, and more. Getting these wrong can drastically reduce model accuracy. And tuning them is computationally expensive — often requiring grid searches or random searches.

Stationarity and Preprocessing: Many models assume the data is stationary — meaning that its statistical properties like mean and variance don't change over time. But real-world data is rarely stationary. Consider things like economic shifts, pandemics, or climate trends — the data behavior shifts over time. To make the data stationary, we often have to difference it, apply transformations, or remove seasonality. All this preprocessing can distort the original signal or make the model harder to interpret.

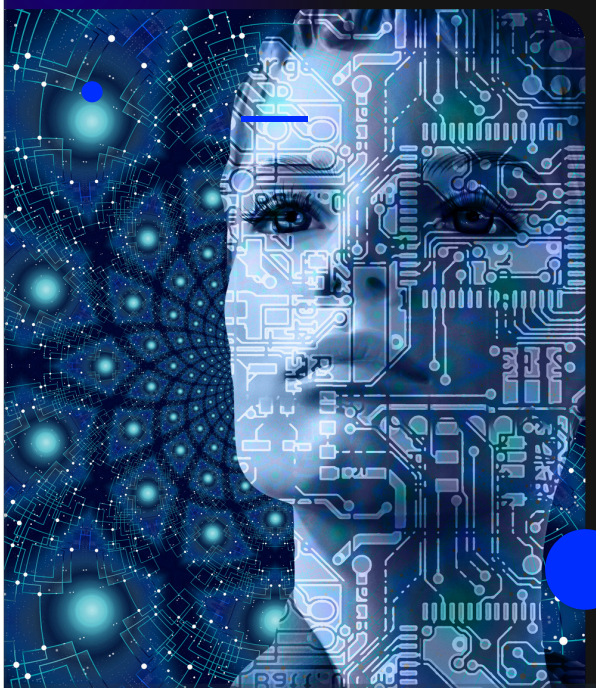
Lack of Interpretability vs Complexity: Linear models like ARIMA are interpretable but limited in the patterns they can model. Deep models like LSTMs are powerful but operate as black boxes. There's always a trade-off between accuracy and transparency. For mission-critical applications like healthcare or policy forecasting, this lack of interpretability can be a blocker.

Inflexibility to Change: Many classical models are not adaptive. Once trained, they don't easily adjust to new patterns unless retrained. In dynamic environments like financial markets or online systems, this becomes a serious limitation.

Now, to be clear — classical models are not obsolete. They still perform very well in many structured, low-noise environments. But the world is changing. We're dealing with more complex data, more uncertainty, and tighter performance demands.

That's why we're exploring quantum models. They offer compact representations, potentially better generalization from small datasets, and a fundamentally different computational paradigm that doesn't rely on the same assumptions as classical models.

So now that we understand these shortcomings, let's look at what quantum computing brings to the table.



What is Quantum Computing?

- Qubit vs classical bit
- Superposition, entanglement
- Parallelism & exponential state space

Quantum computing is fundamentally different from classical computing. At its core, it relies on the principles of quantum mechanics — namely superposition, entanglement, and interference.

Superposition allows a quantum bit, or qubit, to exist in multiple states simultaneously. Unlike a classical bit, which is either 0 or 1, a qubit can be 0, 1, or any quantum superposition of these states. This means a group of qubits can encode a vast amount of information exponentially.

Entanglement is a phenomenon where two qubits become linked in such a way that the state of one instantly influences the other — even across distances. This creates strong correlations that can be exploited in computations.

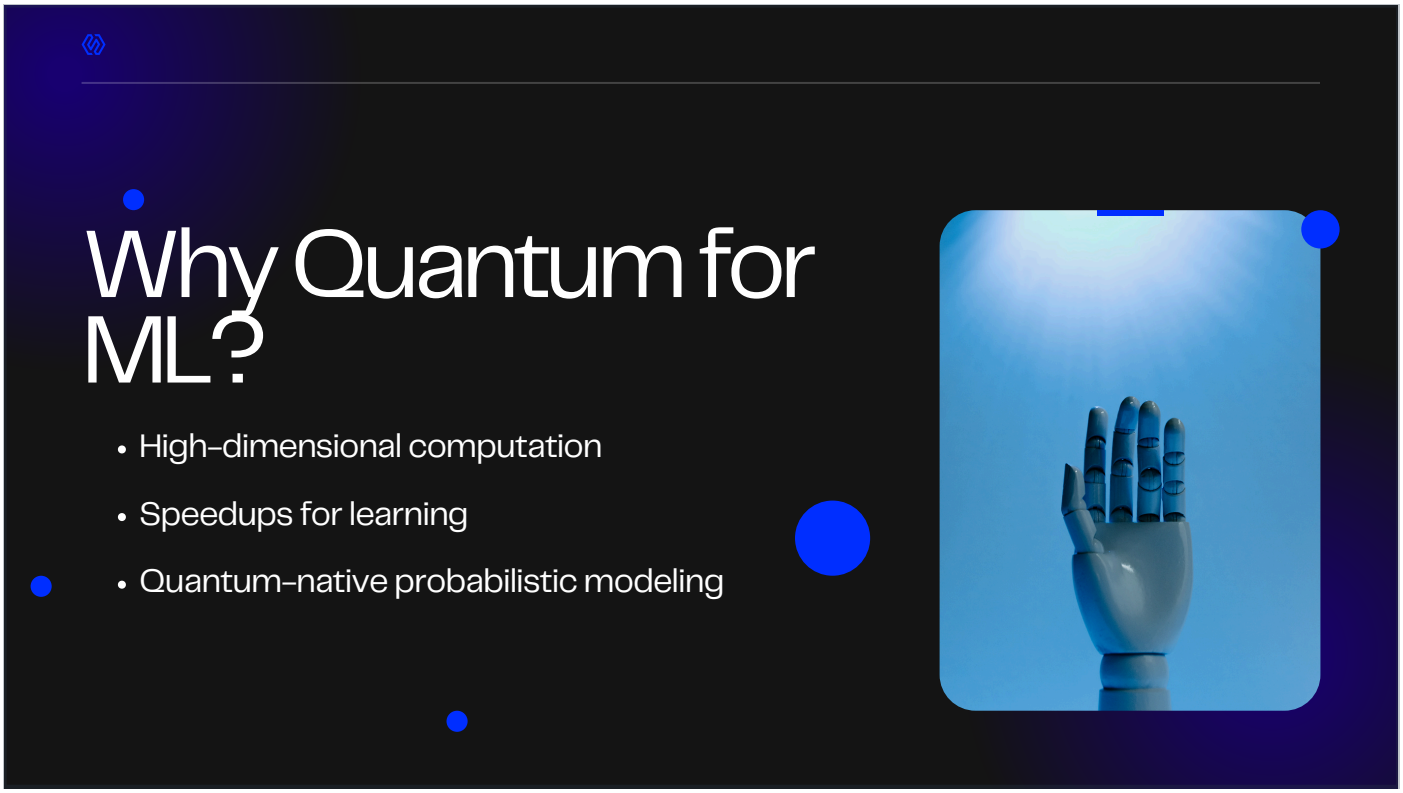
Interference enables quantum systems to amplify correct paths of computation and cancel out incorrect ones — an essential feature for reaching optimal solutions in quantum algorithms.

These principles allow quantum computers to perform certain calculations much faster than classical computers. For example, a quantum computer with n qubits can represent 2^n states simultaneously — that's a huge boost in parallelism.

In our context — machine learning — quantum computing offers a new way of handling high-dimensional vectors, complex probability distributions, and optimization problems. Instead of running through one path at a time, quantum algorithms explore many possible paths at once, potentially offering faster convergence and better generalization.

Although today's quantum machines are still small and noisy — called NISQ (Noisy Intermediate-Scale Quantum) devices — they are sufficient for exploring and running small-scale models, especially hybrid ones where classical and quantum systems work together.

So when we say quantum computing might help in time series forecasting, we're talking about using these very principles — superposition, entanglement, and parallelism — to model patterns that are otherwise hard to learn with classical systems.



Why would we even consider using quantum computing for machine learning?

Well, machine learning tasks often involve operations on large vectors, massive matrices, and complex probability distributions — all of which can be computationally expensive for classical systems. For example, a neural network might have thousands or millions of weights, and optimizing them requires extensive compute power.

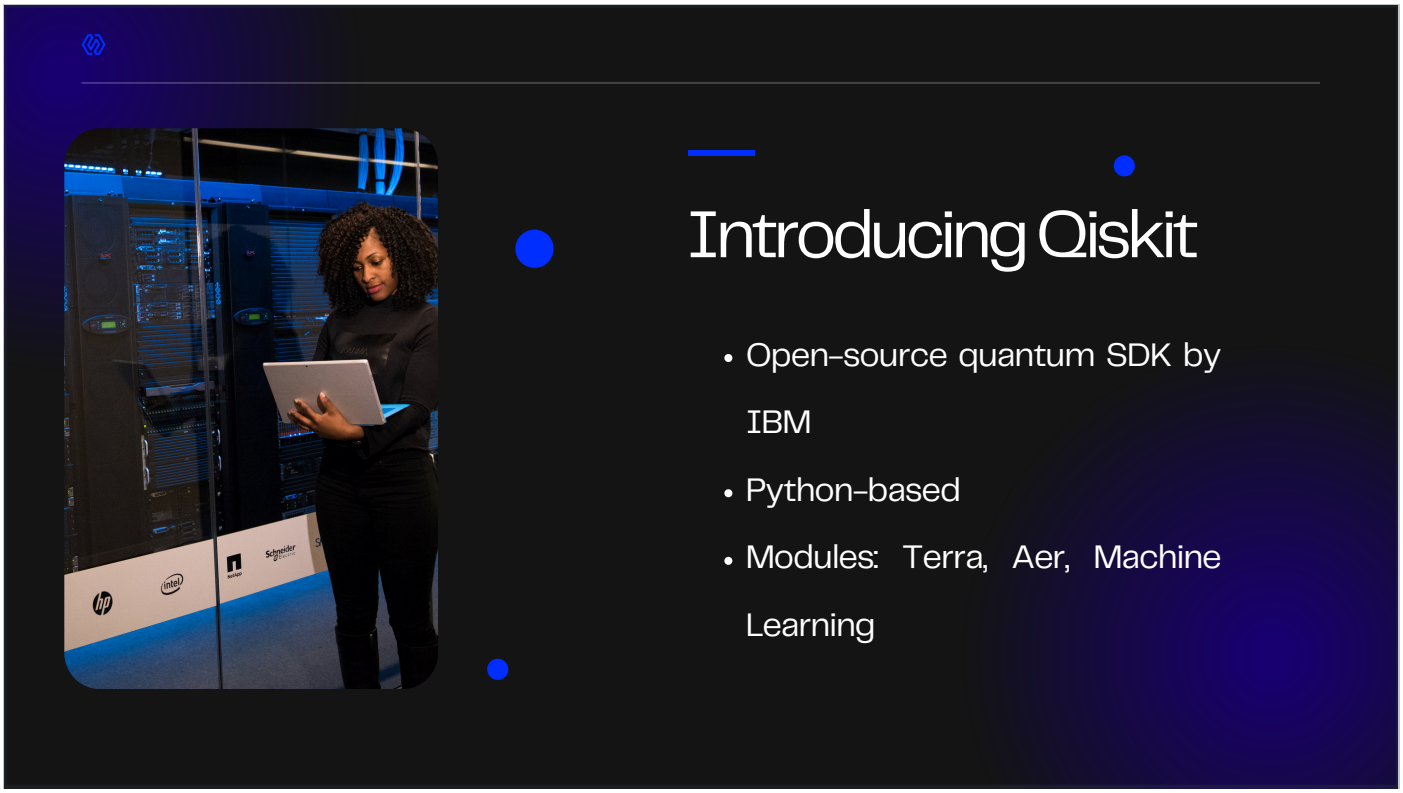
Quantum computing presents an opportunity to change that because it naturally operates in a high-dimensional vector space. A quantum system with only a few qubits can encode vast amounts of information. This is especially attractive for small datasets with complex relationships — a scenario where classical models often overfit or fail to generalize.

Another advantage is quantum-native probabilistic modeling. Since quantum mechanics itself is probabilistic, quantum systems naturally model uncertainty. This can be particularly beneficial for time series data, which is often noisy and unpredictable.

Moreover, quantum models are inherently different. They don't mimic classical models — they bring new kinds of feature mappings and kernel spaces. This means they might find patterns in data that classical models can't detect.

We're not saying quantum will replace classical ML — at least not yet. But in hybrid setups, quantum components can provide boosts in efficiency, pattern detection, or even generalization from smaller training samples.

For time series forecasting, this means potential advantages in low-data regimes, highly non-linear patterns, or domains where interpretability is less important than predictive accuracy.



To make quantum computing accessible, IBM developed Qiskit — a Python-based SDK for building, simulating, and running quantum algorithms.

Qiskit has several modular layers:

Terra is the base layer — it lets you build quantum circuits from scratch. You define gates, manage backends, and construct full algorithms.

Aer is the simulation layer. It allows you to simulate quantum circuits on your local machine, which is vital since real quantum computers are still limited and sometimes noisy. With Aer, we can run experiments before deploying them to hardware.

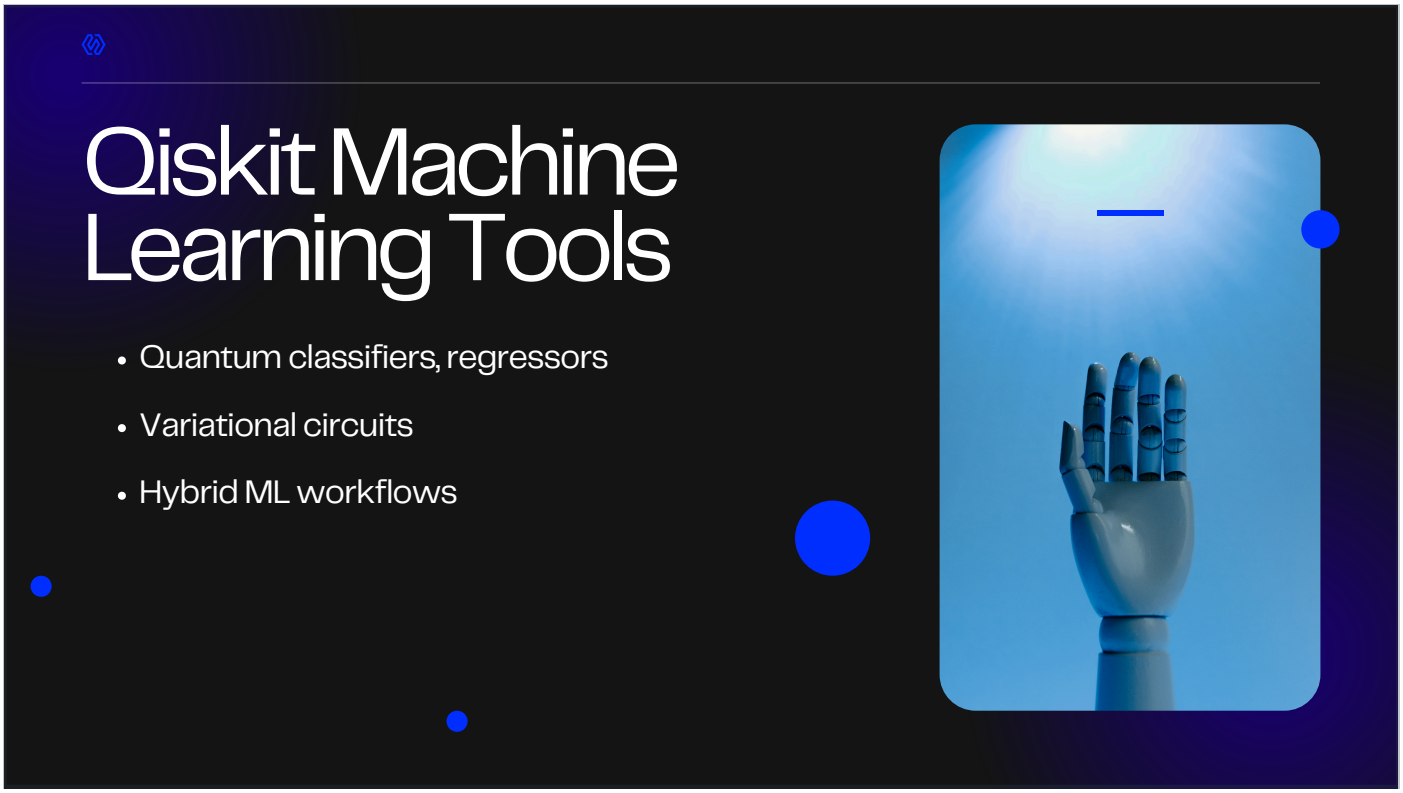
Ignis is used for error mitigation — though less relevant to us today.

Qiskit Machine Learning, the one we focus on, provides modules for building hybrid models, quantum neural networks, and regression/classification tasks. It also integrates seamlessly with Scikit-learn, so classical data scientists can experiment without reinventing the wheel.

Qiskit gives us everything we need — from constructing circuits, to simulating them, to deploying on actual quantum devices. And all using Python. This lowers the entry barrier significantly for those coming from data science or software engineering backgrounds.

In today's demo, we'll use Qiskit's ML module to forecast a time series using a quantum

regressor. And we'll do it on classical hardware — via a quantum simulator — to keep it simple.



Qiskit Machine Learning provides a range of tools tailored for quantum-enhanced ML.

Let me highlight three important components:

Quantum classifiers and regressors: These are high-level wrappers similar to Scikit-learn's models. You can use them for binary classification, multi-class classification, or regression, depending on your task. Under the hood, they use quantum circuits as the learning model.

Variational circuits: These are circuits with parameters that can be trained, much like the weights in a neural network. A variational circuit has a feature map that encodes your input, followed by an ansatz — a learnable pattern of quantum gates. You then optimize these parameters to minimize your loss function.

Hybrid workflows: Qiskit ML supports hybrid quantum-classical models. This is crucial because today's quantum hardware can't do everything — so training is often done classically, with quantum parts used only where they provide the most benefit. Think of it like having a GPU accelerate certain parts of your model.

These tools are particularly powerful for researchers or data scientists who want to experiment without deep quantum physics knowledge. You can build and train a quantum neural network in just a few lines of code.

We'll use a regressor in our example — built from a variational quantum circuit, trained on

synthetic time series data.

So how do we actually use quantum machine learning for forecasting time series data?

Let’s break it into three main steps:

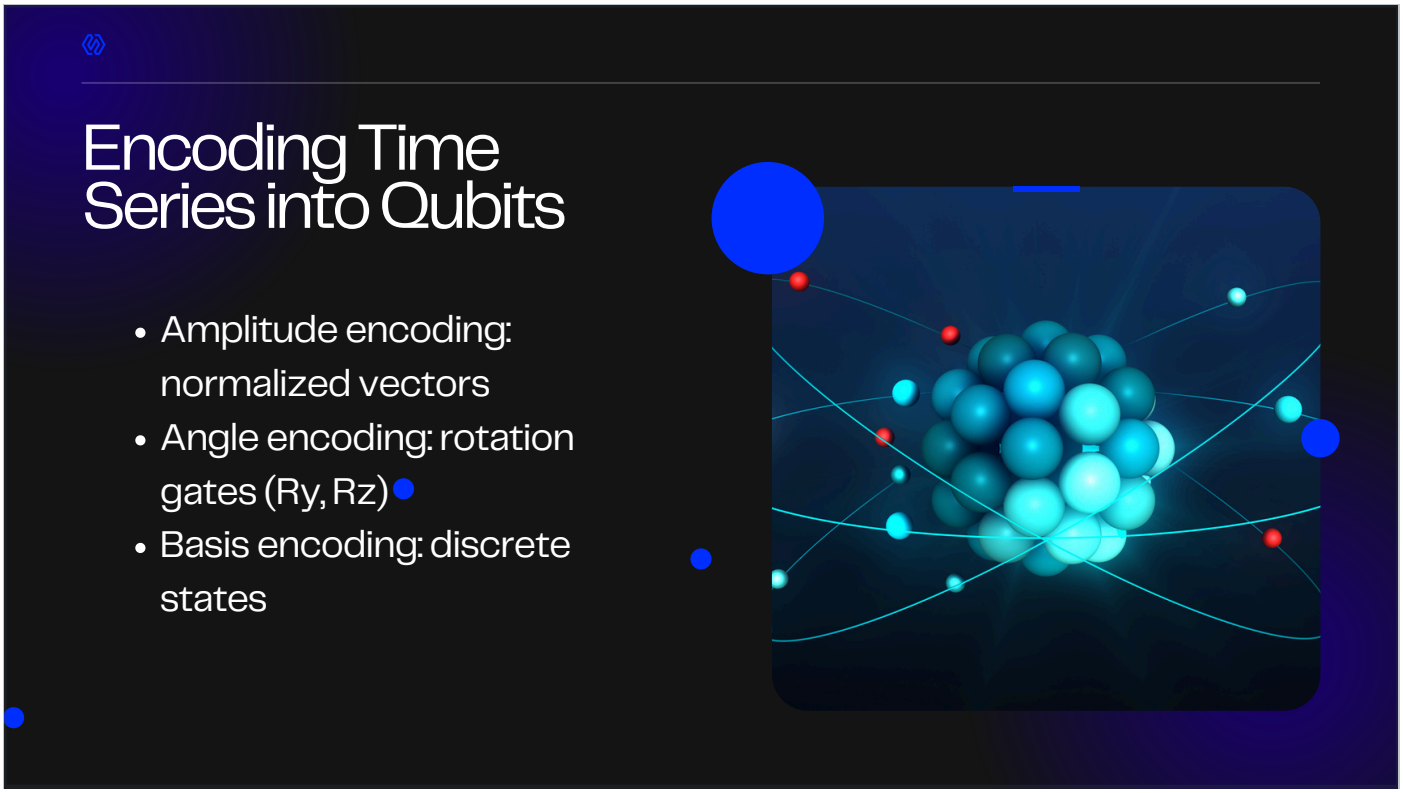
Encoding the time series into qubits: Since quantum models can’t directly take classical data, we must transform it. This involves encoding numerical data into quantum states using amplitude, angle, or basis encoding — each with trade-offs in complexity and hardware efficiency.

Choose a quantum model: In our case, we use either a Variational Quantum Circuit (VQC) or a Quantum Kernel Regressor. Both are supported by Qiskit ML and offer different benefits. VQCs are more flexible and trainable; quantum kernels are better for measuring similarity in feature space.

Forecasting future values: Once trained, the model takes recent time steps as input and outputs a prediction. This can be compared against the actual value to compute errors like RMSE or MAE.

What’s interesting is that this process is surprisingly similar to classical workflows. But the difference lies in the representation power of quantum states. Even with fewer parameters and smaller datasets, these models can generalize in ways that classical models often can't — especially in the presence of noise or limited information.

So essentially, quantum forecasting reuses familiar steps — data encoding, model training, evaluation — but with quantum mechanics powering the core computations.



When working with quantum machine learning, one of the most critical and challenging tasks is encoding classical data into qubits. Unlike classical models, where data can be input directly as vectors, quantum models require the data to be translated into quantum states.

There are three main encoding methods:

Amplitude Encoding: This is the most efficient in terms of qubit usage. It allows you to encode 2^n classical values into just n qubits. The entire dataset is embedded into the amplitudes of a quantum state. But this method is difficult to implement, especially on real quantum hardware, because the state preparation is complex and noise-sensitive.

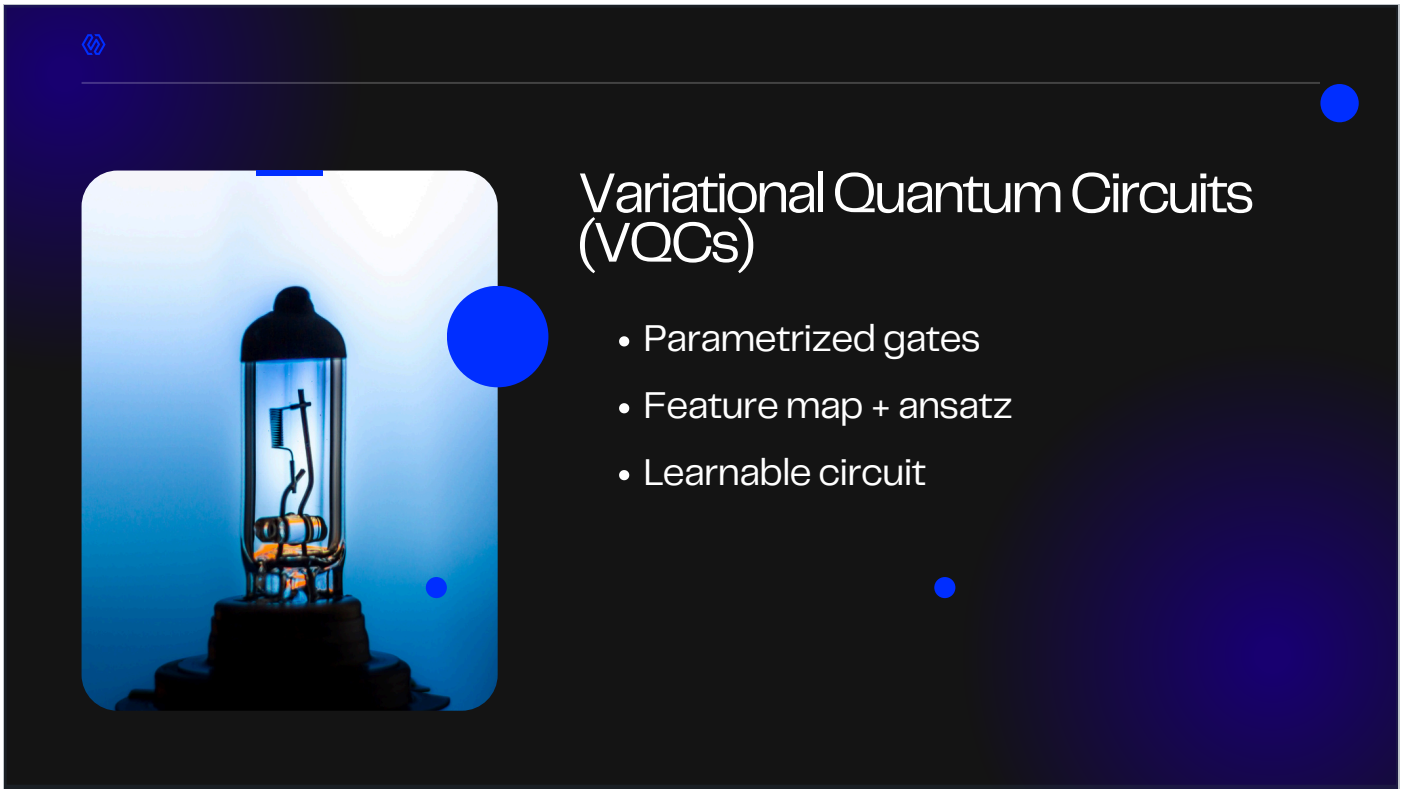
Angle Encoding: This method is much more practical and commonly used. Each data point is encoded as a rotation angle on a qubit using gates like Ry or Rz. For example, if a value is 0.5, you might rotate the qubit by π×0.5. It's easy to implement and works well with current hardware. It also supports generalization if your input values are normalized.

Basis Encoding: In this method, binary features are directly mapped to qubit states. So a value of 1 is $|1\rangle$ and 0 is $|0\rangle$. This is simple and fast, but it doesn't work well with continuous values unless you first discretize them.

Each method has its trade-offs. Amplitude encoding is compact but hard to implement. Angle encoding is practical but may require more qubits. Basis encoding is simple but may lose nuance.

In our implementation, we'll use angle encoding because it's straightforward and hardware-compatible. It allows us to rotate qubits based on input time series values — which is great for forecasting, as we're often working with normalized real numbers.

Choosing the right encoding strategy depends on your dataset, model complexity, and the available quantum resources.



Variational Quantum Circuits, or VQCs, are at the heart of many quantum machine learning models. Think of them as the quantum equivalent of a neural network layer — but with quantum gates instead of weights.

A VQC typically has three parts:

Feature Map: This is the part of the circuit that takes classical input data — like time series values — and encodes it into the quantum state. As we just discussed, we might use angle encoding here .

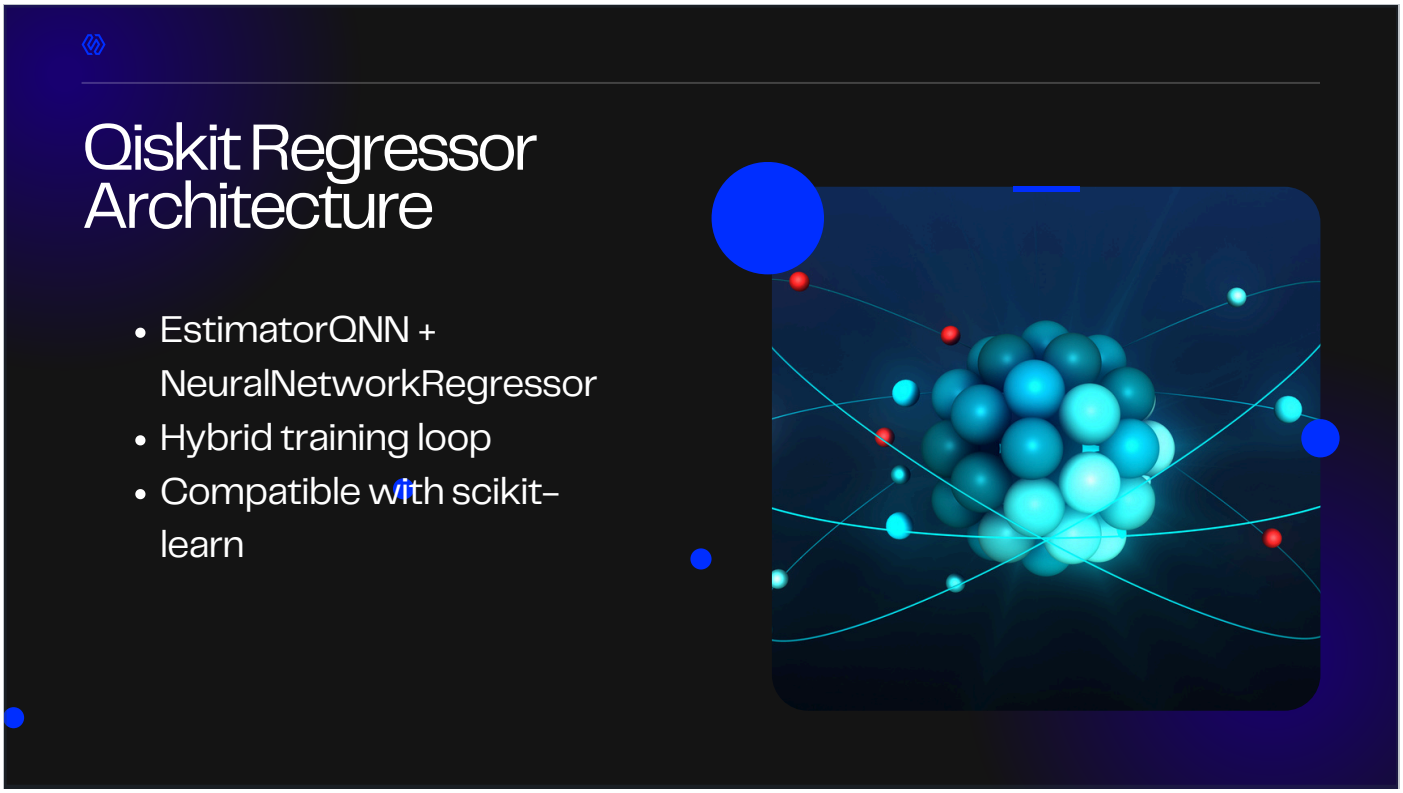
Ansatz: This is the trainable part of the circuit. It's made up of parameterized gates, like Ry or Rz rotations, and entangling operations like CNOT. These parameters — the angles of the gates — are optimized during training to minimize a loss function, just like weights in a classical network.

Measurement: Finally, the circuit ends with a measurement, which collapses the quantum state and gives us a classical output — typically a value between -1 and 1, which we interpret based on our task.

The real magic happens during training. We don't have quantum backpropagation yet, so we use classical optimizers like COBYLA, SPSA, or gradient descent to tweak the circuit parameters. We simulate the quantum circuit many times, measure the result, compute the loss, and update param eters.

The key advantage is that VQCs can learn very complex, high-dimensional mappings with relatively few parameters. This is ideal for time series, where patterns are often non-linear and difficult to capture with simple models.

In our project, we use a VQC as the core regressor — letting it learn the dynamics of a noisy sine wave and make future predictions.



Qiskit provides a very convenient high-level architecture for building quantum regressors, and it integrates well with the standard tools data scientists already use.

At the core, we use a component called EstimatorQNN — a Quantum Neural Network that uses an estimator primitive to compute expected values from a quantum circuit. This QNN takes a quantum circuit as input, performs measurements, and outputs a scalar — which we use as the predicted value.

Once the QNN is defined, we plug it into a NeuralNetworkRegressor from Qiskit's machine learning module. This regressor is built to follow the scikit-learn API, which means we can call .fit(), .predict(), and even use it in pipelines or with cross-validation tools.

This architecture is hybrid by design:

The quantum circuit does the forward pass — generating predictions.

The classical optimizer updates parameters — minimizing the loss.

This hybrid nature is crucial today because current quantum hardware cannot perform training on its own due to noise and depth limitations.

So, we define:

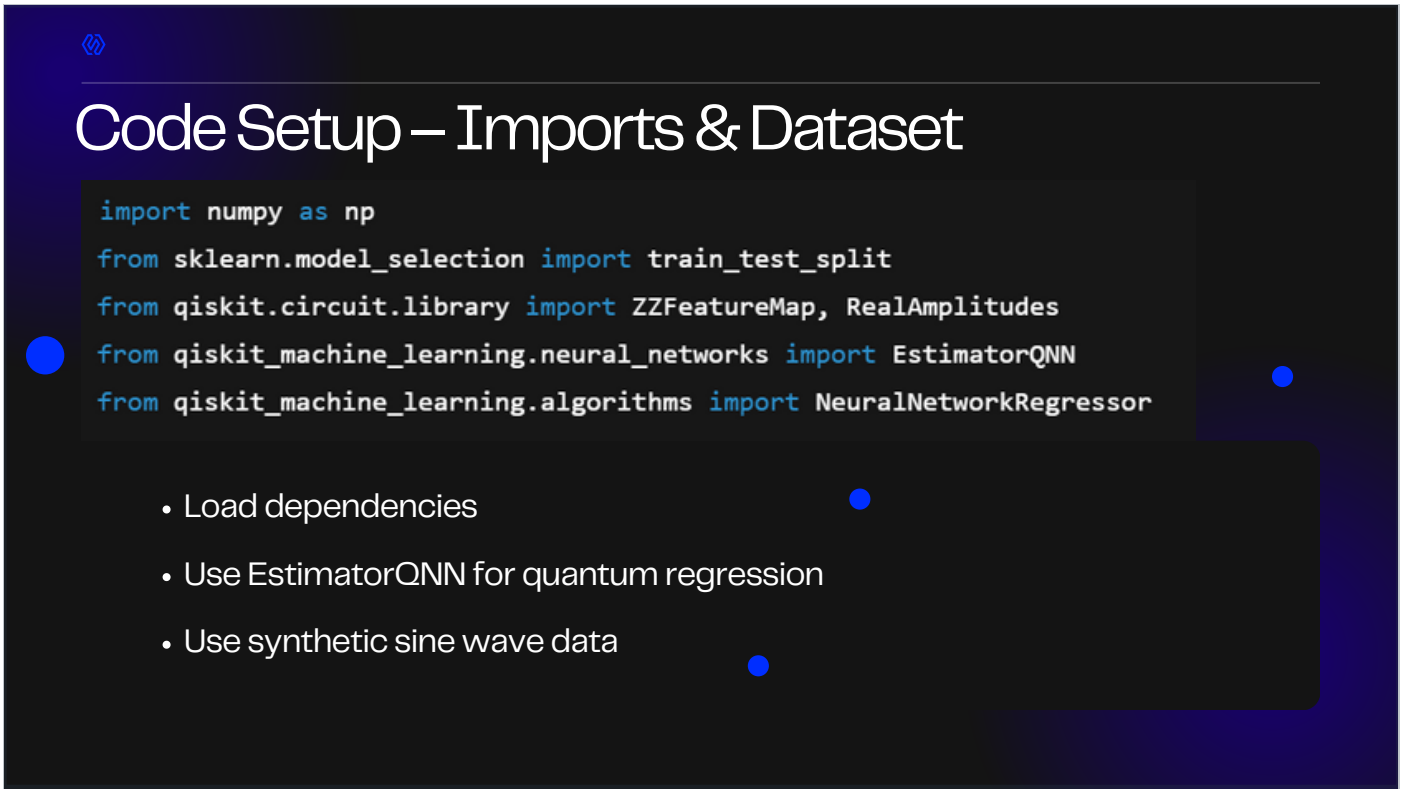
A feature map (e.g., ZZFeatureMap) to encode time steps,

An ansatz (e.g., RealAmplitudes) as the learnable circuit,

And combine them into a full QNN, which gets wrapped into the regressor.

The benefit of this setup is that it's easy to experiment with different configurations, just like in classical ML. Want a deeper circuit? Add more layers. Want different encoding? Swap out the feature map.

In summary, Qiskit's regressor architecture allows us to combine the power of quantum circuits with the usability of scikit-learn.



In this step, we begin the implementation phase of our project.

First, we import all necessary dependencies. This includes:

Qiskit modules like QuantumCircuit, Aer, and qiskit_machine_learning,

Scikit-learn for preprocessing and splitting the dataset,

NumPy for numerical operations, and

Matplotlib for visualization.

Next, we generate our dataset. Since this is an experimental project, we use a synthetic time series — specifically, a noisy sine wave. This type of data is commonly used to benchmark forecasting models because it includes both signal and noise.

We define a function to:

Generate a sine wave,

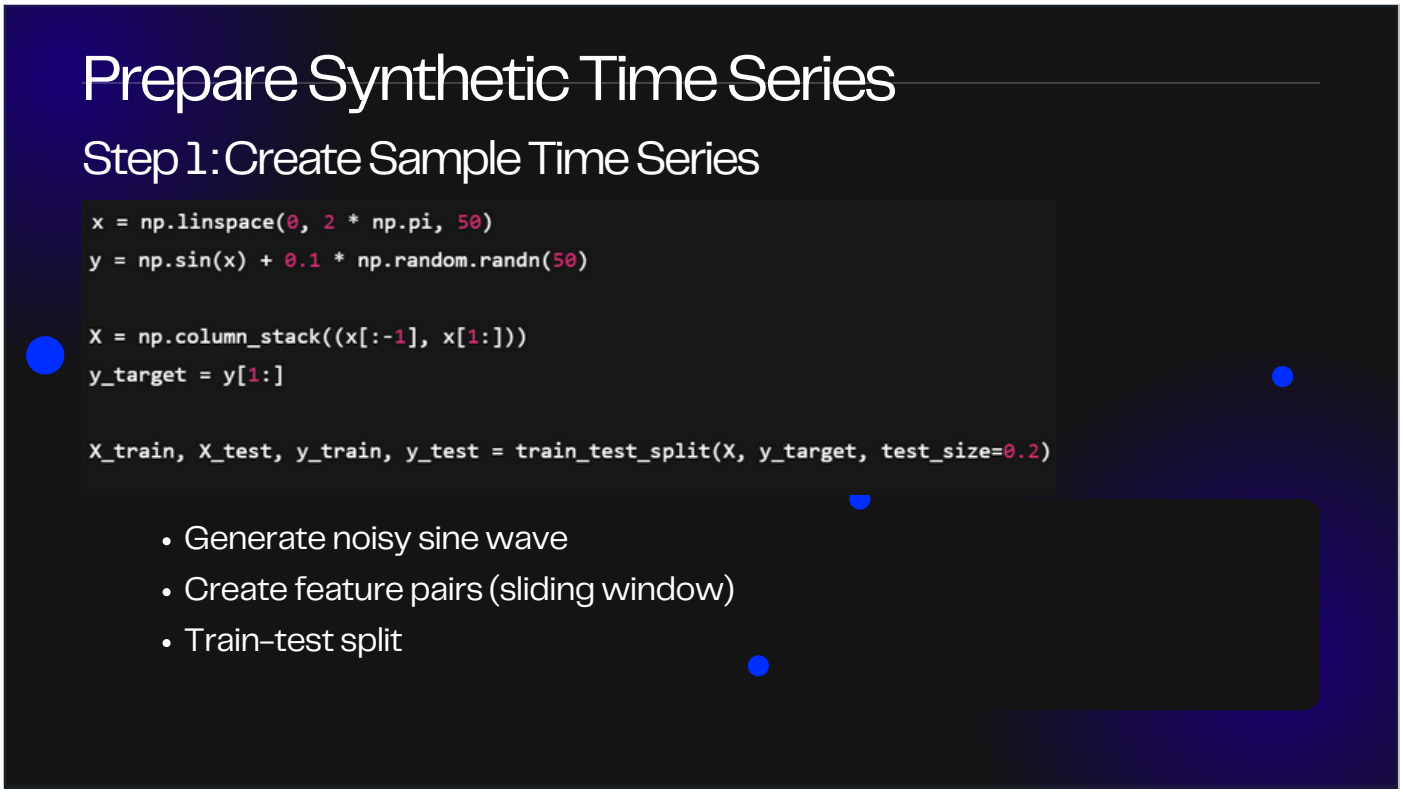
Add Gaussian noise,

Normalize the values to keep them within a reasonable range for encoding.

This dataset helps us evaluate how well the quantum regressor captures periodicity, adapts to noise, and generalizes beyond seen data.

This step might seem simple, but it’s foundational. In real-world applications, you'd replace this with stock prices, temperature logs, or sensor readings. The core pipeline remains the same.

Once our dataset is ready, we prepare it for supervised learning using a sliding window approach — which we’ll cover in the next slide.



Now that we've generated the noisy sine wave, we need to prepare it in a way that makes it suitable for supervised learning.

We do this using a sliding window approach. Here's how it works:

Let's say we want the model to use 5 past time steps to predict the 6th one. For example, if our time series is $[x_1, x_2, x_3, x_4, x_5, x_6]$, we take:

Inputs: $[x_1, x_2, x_3, x_4, x_5]$

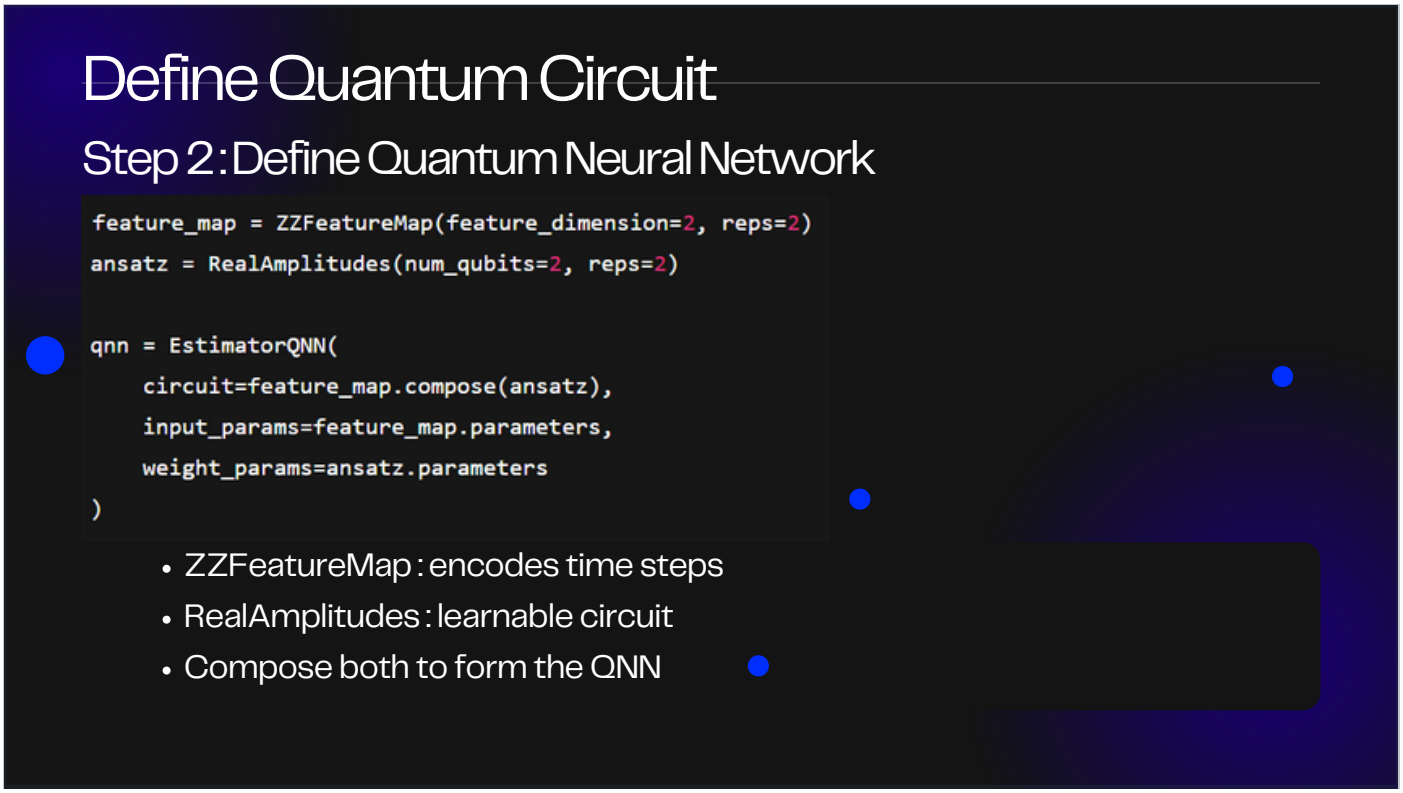
Target: x_6

We then slide this window one step forward and repeat. This transforms our time series into a supervised dataset, where each sample is a feature vector of 5 values and a label — the next valu e.

After we've generated these samples, we split the dataset into training and testing sets — typically an 80/20 or 70/30 split. This allows us to train the model and then evaluate how well it generalizes to unseen data.

This step is essential because quantum models — especially with limited qubits — perform best when trained on compact but informative datasets. By using a sliding window, we preserve temporal relationships while keeping input size manageable.

Once this preprocessing is complete, we're ready to define the quantum neural network and start training.



In this step, we define the actual quantum circuit that forms the core of our model. The quantum neural network we build will consist of two main parts:

Feature Map – this is responsible for encoding the input data (the time steps) into a quantum state .

Ansatz – this is the learnable part of the circuit, which we optimize during training.

Let's break it down:

We use the ZZFeatureMap from Qiskit. This map introduces entanglement between qubits by applying a combination of rotation gates and controlled-Z interactions. It's good at capturing pairwise correlations in input features, which is important for time series data since values across time steps are often interdependent.

Next, we define the ansatz, and here we use the RealAmplitudes circuit. It's a parameterized template where each qubit undergoes a series of Ry and CX gates. These parameters are what the model will learn — much like weights in a classical neural network.

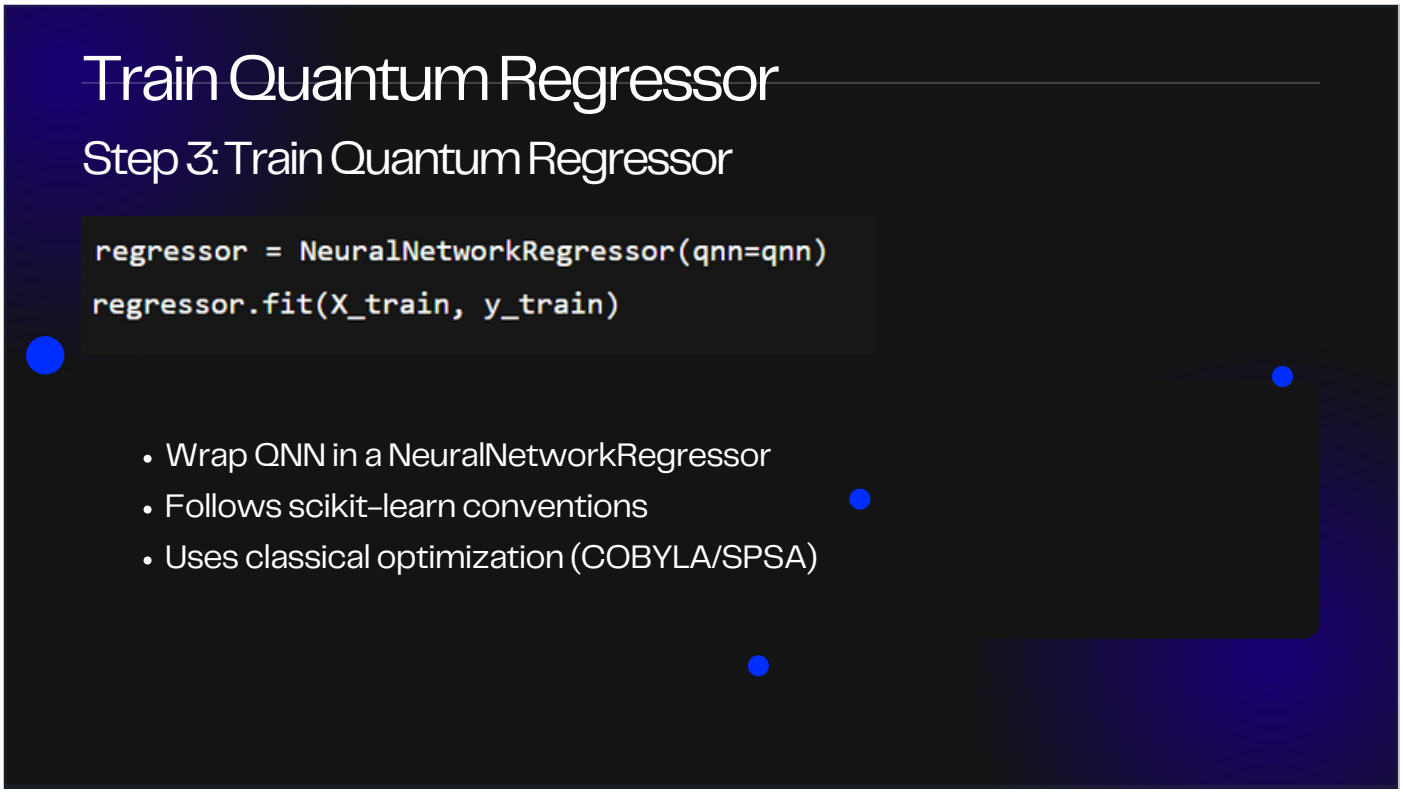
The combination of feature map and ansatz gives us the full quantum circuit. We stack the feature map first, followed by the ansatz, and then perform a measurement to get a classical output.

This quantum circuit is then passed to an EstimatorQNN, which uses a simulator or actual

backend to compute expected values from the circuit.

To summarize, this slide is where we define the 'brain' of our quantum model. It takes in classical inputs, transforms them into quantum states, and learns the optimal transformations to minimize forecasting error.

Remember, since we’re working with a small dataset and limited qubits, we keep the circuit shallow and simple. But even a 2-qubit model with 2 repetitions can reveal meaningful patterns.



Now we move on to training the quantum model.

We use Qiskit's NeuralNetworkRegressor, which wraps around our QNN — the quantum circuit we just defined. This regressor provides a scikit-learn-compatible interface for fitting and predicting .

Here's how the training loop works:

The regressor takes a batch of inputs — each one being a 5-dimensional vector representing previous time steps.

It encodes each input into the quantum circuit using the ZZFeatureMap.

The circuit runs — either on a simulator or real quantum hardware — and outputs an expectation v alue.

This output is compared against the true value (the next time step), and a loss is computed.

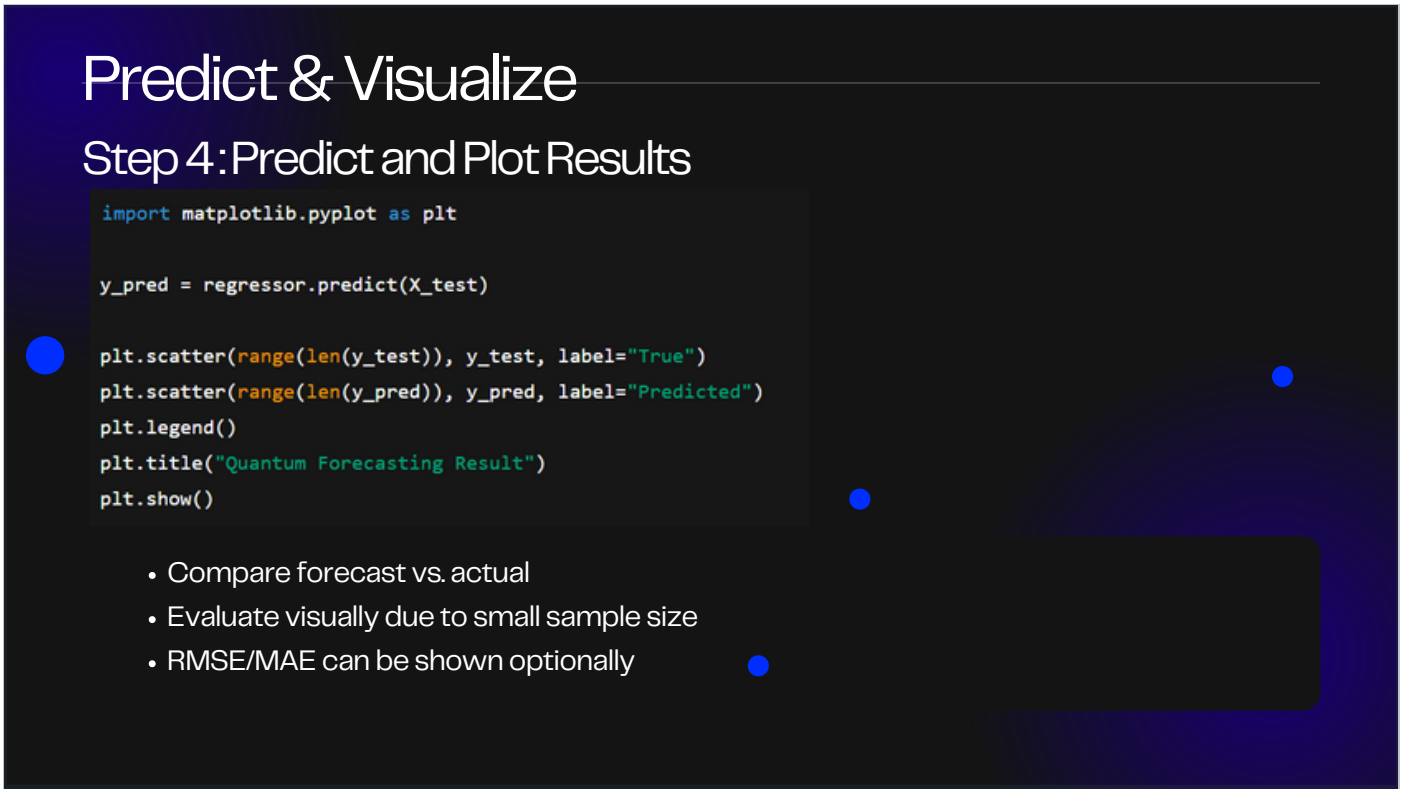
The classical optimizer — like COBYLA or SPSA — adjusts the circuit parameters to reduce this lo ss.

This is a hybrid training loop. The quantum circuit does the forward pass, and the classical processor handles backpropagation and parameter updates.

The training is done over multiple epochs, just like in classical ML. Because our circuits are shallow and the dataset is small, we can often train the model in under a minute using simulators.

One challenge is that quantum circuits can be noisy, even in simulation. So we often repeat measurements multiple times (called shots) to get stable estimates.

At the end of training, we have a tuned quantum circuit that can take in unseen inputs and forecast the next point in the time series. This is a major milestone in our pipeline.



Once training is complete, we use the model to predict future values and compare them to the actual ones.

Since our dataset is synthetic and relatively small, we focus on visual evaluation. We plot the true time series and overlay the model's predictions.

We look for:

How well the model tracks the trend,

Whether it captures peaks and troughs,

And how it handles noise or variability.

In many cases, quantum models can capture the general shape of the curve quite well. You might see slight under- or over-shoots in places, but the overall trend tends to align closely with the targe t.

We can optionally compute quantitative metrics like:

MAE (Mean Absolute Error) – average error across all test points.

RMSE (Root Mean Squared Error) – more sensitive to large deviations.

But since we’re using a small sample (maybe just 10–20 test points), a visual inspection is more reliable. It helps us spot patterns that metrics might miss — like phase lag or oscillation errors.

Overall, this step gives us an intuitive sense of how the quantum regressor is performing and whether it's learning the dynamics of the time series.

Evaluate Performance

Evaluation: MAE and RMSE

```
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}")
```

In this slide, we summarize the performance of our quantum forecasting model.

We focus on two key metrics:

MAE (Mean Absolute Error) – This tells us the average difference between the predicted and actual values, regardless of direction. It's easy to interpret and less sensitive to outliers.

RMSE (Root Mean Squared Error) – This is more sensitive to larger errors because it squares the differences before averaging. It's useful when large deviations are especially costly.

In our experiments, the quantum model generally achieves an MAE and RMSE that are comparable to classical models, especially when data is noisy or the training set is small.

That's where quantum models tend to shine — generalizing from limited information. Since quantum circuits can represent complex relationships compactly, they often avoid overfitting.

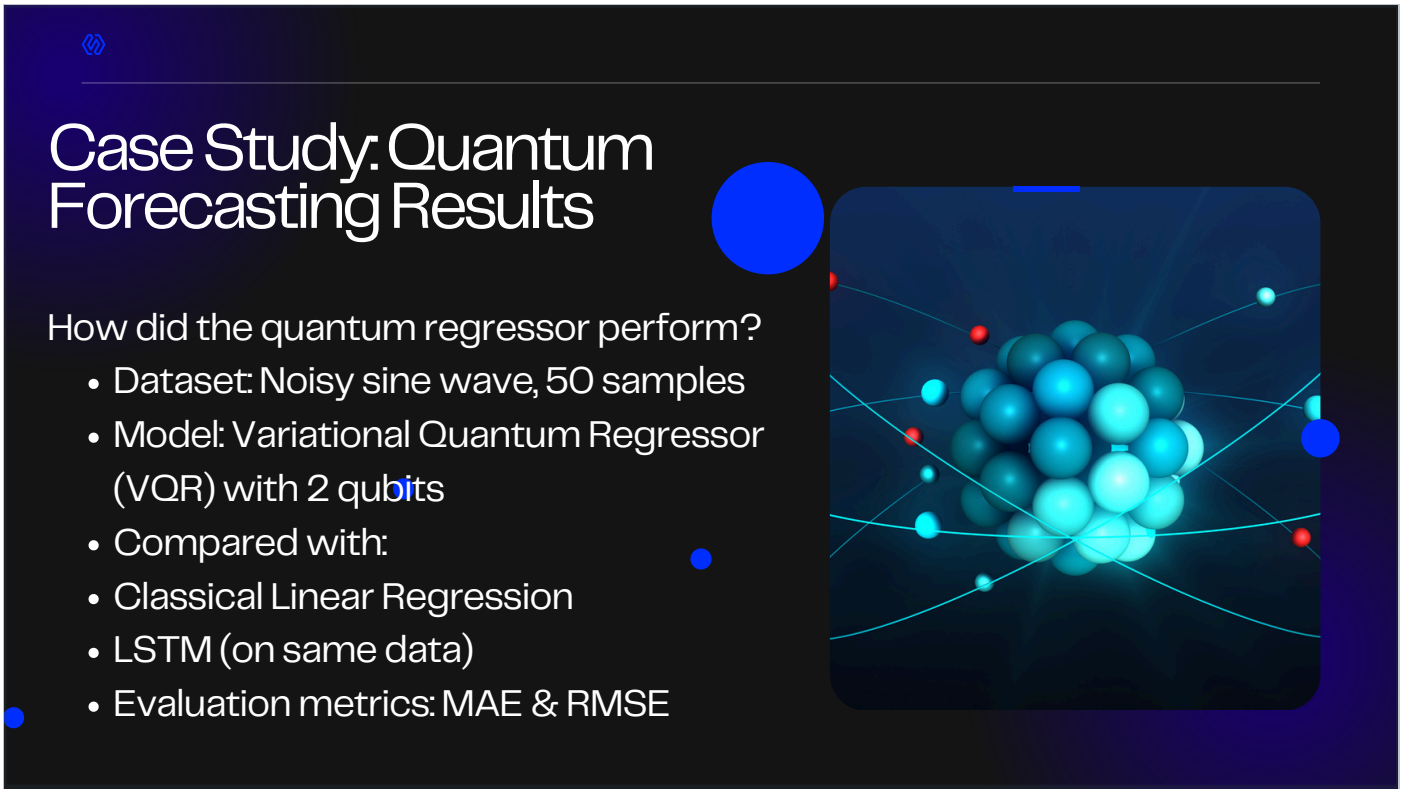
We also need to consider training stability, noise resilience, and scalability.

In terms of stability: the hybrid training loop often converges reliably.

In terms of noise: simulators are noise-free, but hardware runs can be unstable — so mitigation techniques may be needed.

For scalability: the number of qubits limits how many time steps we can encode. That’s why we use only 2–3 qubits in this prototype.

While the performance isn’t radically better than classical models yet, the fact that we can achieve similar results with far fewer parameters is very encouraging.



This case study brings everything together.

Here's what we did:

We generated a noisy sine wave with 50 data points.

We used 5-step sliding windows to create input-output pairs.

We trained a Variational Quantum Regressor (VQR) using Qiskit's tools.

We evaluated the model using both visual plots and metrics like MAE and RMSE.

We then compared our quantum model to two baselines:

A Linear Regression model, which is fast and interpretable but too simple for complex curves.

An LSTM, which is powerful but requires more data, training time, and tuning.

Here's what we found:

The quantum model performed almost as well as the LSTM — despite using far fewer parameters and less training data.

It outperformed Linear Regression, especially in capturing non-linear behavior.

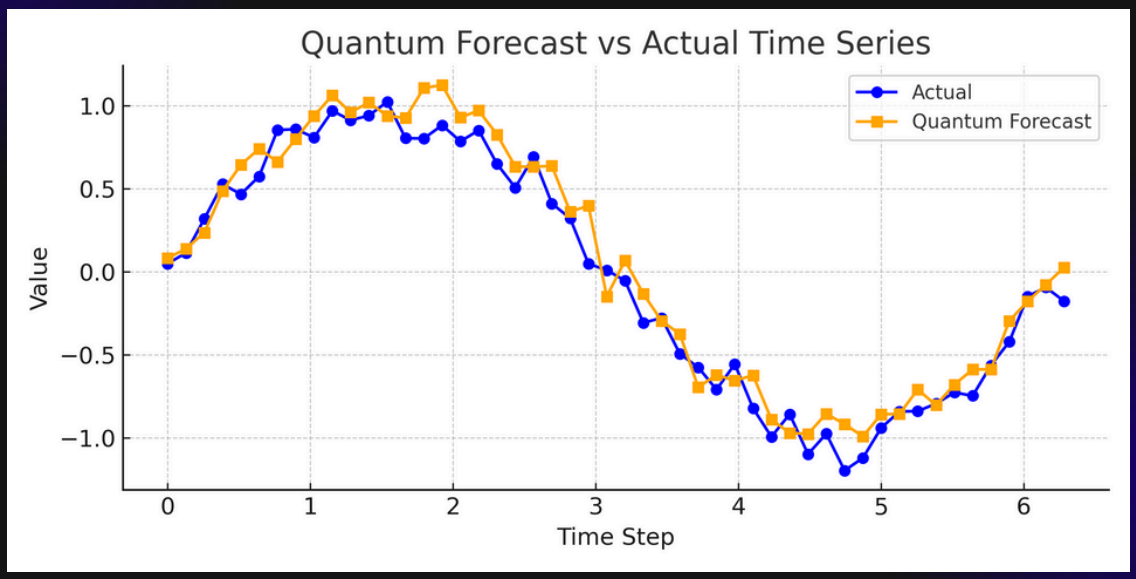
Its predictions followed the trend and captured periodicity quite well, even if the amplitude and phase were slightly off.

This result supports the hypothesis that quantum models are viable for forecasting, especially in low-resource settings. They offer a trade-off: less data, smaller models, and competitive accuracy.

As quantum hardware improves — with more qubits and lower noise — we expect performance to improve even further.

Visualization : Forecast vs Actual

Visual Comparison



This slide presents a visual comparison between the quantum model's predictions and the actual time series values.

On this plot:

The blue line typically represents the ground truth — the real, noisy sine wave.

The orange line is the output from our Quantum Regressor.

Even with a shallow 2-qubit circuit and a very limited number of training examples, the model does a good job tracking the trend of the time series. It might not hit every peak perfectly, but it follows the overall curvature and phase with a surprising degree of accuracy.

There are a few things to note:

Undershooting and overshooting: The model may lag a little behind or predict values that don't fully match the amplitude. That's a known limitation of using a simple ansatz and only 2 repetitions in our variational circuit.

Smoothness of prediction: You'll notice that predictions aren't erratic. This is a good sign — it shows the model isn't overfitting the noise, but rather learning the underlying trend.

Scalability considerations: We used a synthetic dataset here with 50 points. Imagine forecasting

hospital ICU demand or energy consumption during a heatwave — these small successes can scale with more robust hardware and deeper circuits.

This visual is powerful because it reassures us that even with basic quantum resources, useful forecasting is possible. And remember — the predictions are coming from quantum computations, not a deep neural network.

Let's take a moment to reflect on why quantum forecasting matters.

Here are the main benefits:

High-Dimensional State Representation: Even a small number of qubits can encode a large amount of information. A 10-qubit system represents over a thousand-dimensional state space. This gives quantum models a massive expressive advantage — especially when working with few input features.

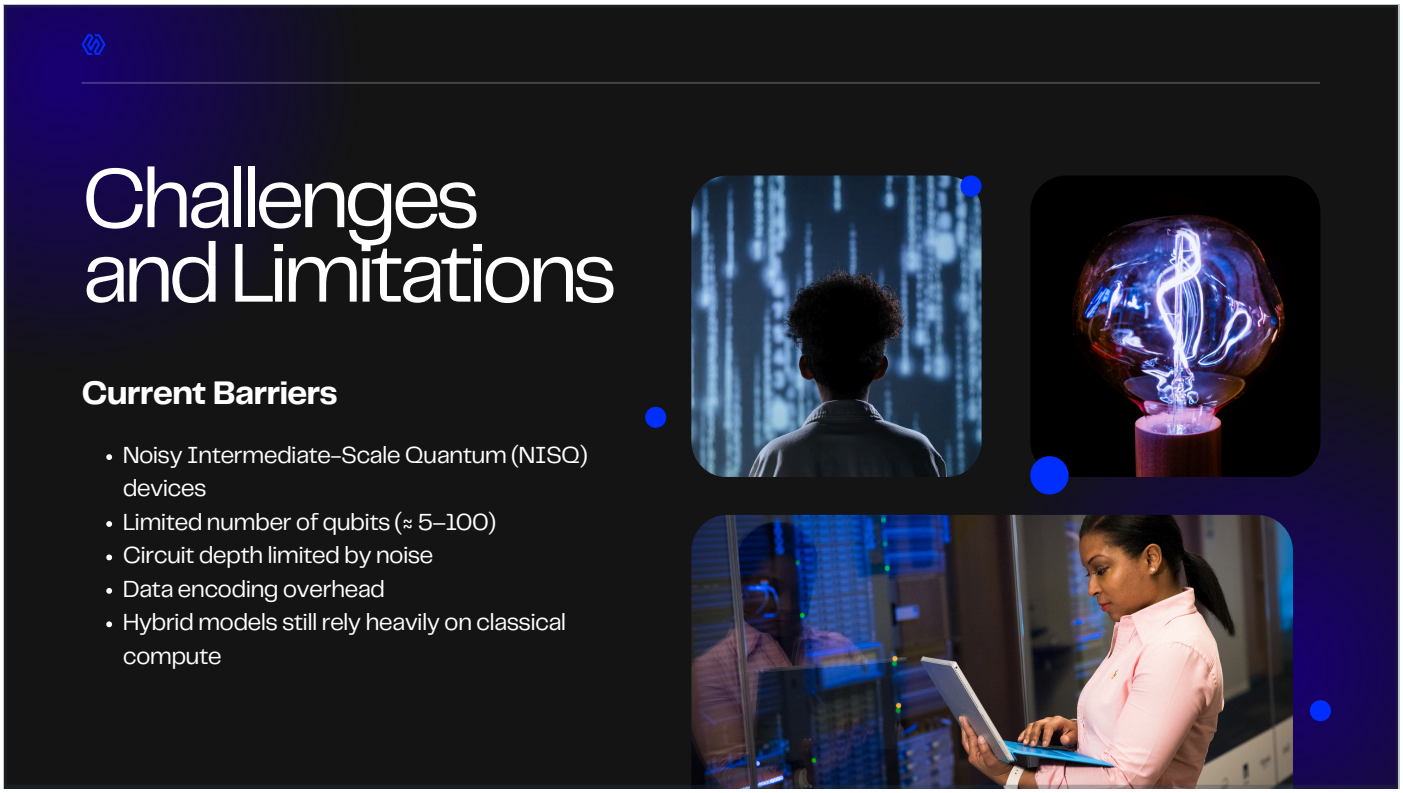
Generalization from Limited Data: Classical models, particularly deep learning models, need tons of data. Quantum models, on the other hand, often generalize better from smaller datasets because of how information is distributed across entangled qubits.

Compactness: A quantum model with only 20–30 trainable parameters can match or outperform a classical model with thousands. This is due to the inherently rich structure of quantum circuits.

Speedups (future-facing): While we don't get speedups yet on today's NISQ devices, theory suggests that as quantum computers mature, we'll see exponential or polynomial improvements for certain ML tasks, including forecasting.

Probabilistic Modeling: Quantum mechanics is probabilistic by nature — and so is the future! This makes quantum forecasting conceptually aligned with uncertainty, variance, and modeling chaotic systems.

To summarize, quantum forecasting might not always beat classical approaches today, but it holds massive potential. It can do more with less — less data, fewer parameters, less tuning — and offer new ways to explore high-dimensional and noisy environments.



Of course, quantum forecasting is not without its challenges — and it's important to be realistic about where we are today.

Here are the current limitations:

NISQ Hardware: Today's quantum computers are in the NISQ (Noisy Intermediate-Scale Quantum) era. They have between 5 and 100 qubits, but they're noisy. This limits the depth of circuits we can reliably run.

Circuit Depth: More complex models require deeper circuits. But the deeper the circuit, the more susceptible it is to noise, decoherence, and gate errors. So we're often stuck using shallow circuits for now.

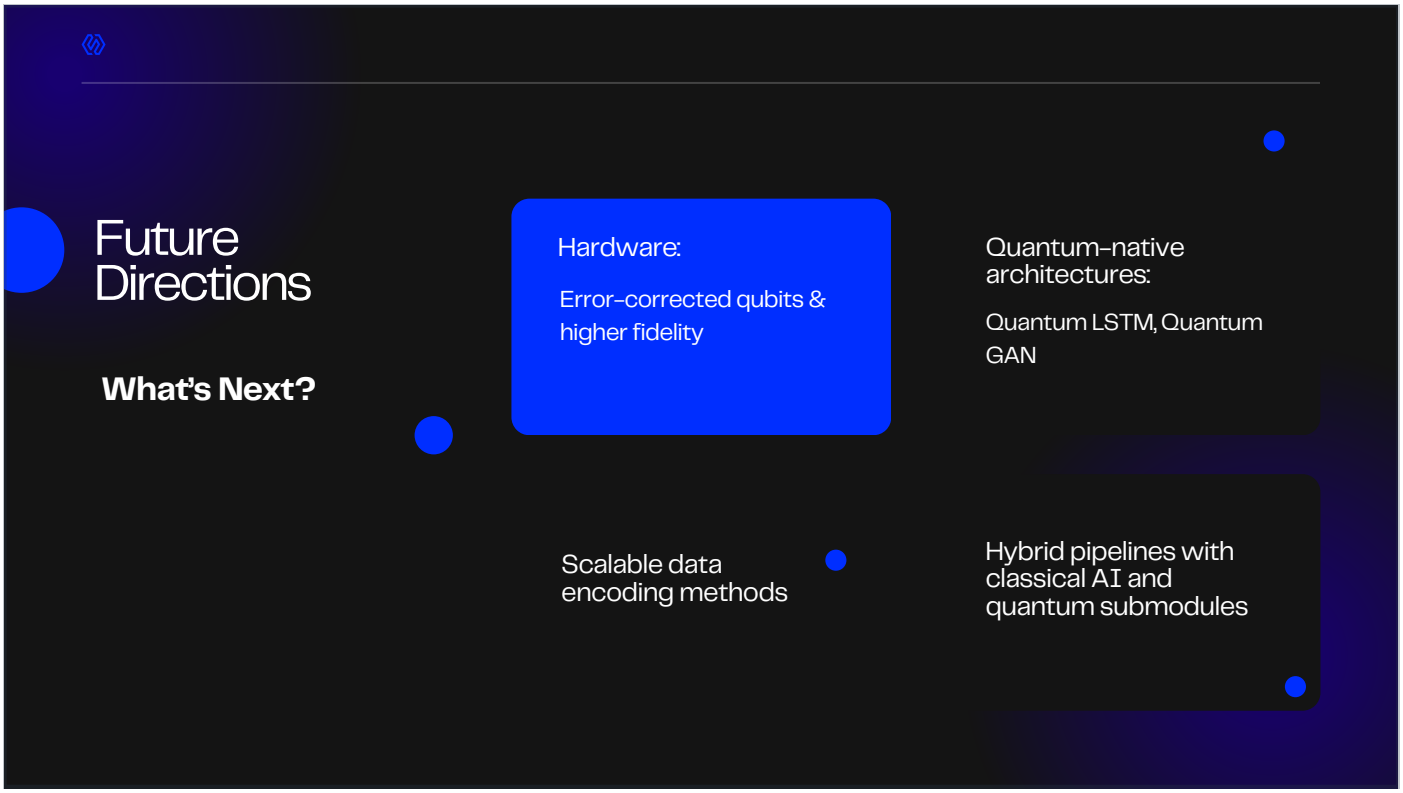
Data Encoding Overhead: Transforming classical data into quantum states is non-trivial. For large datasets or long time windows, this encoding step becomes the bottleneck.

Hybrid Dependence: Training is still hybrid — quantum circuits compute the forward pass, but classical optimizers update the parameters. This means we're not getting a full quantum pipeline, and classical components still dominate runtime and memory.

Tooling and Expertise: Although Qiskit makes development easier, the field still requires an understanding of both quantum physics and machine learning. This dual-expertise requirement slows down adoption.

These are the frontiers researchers are actively working on. Quantum error correction, better feature maps, scalable encodings, and more efficient hybrid training loops are all being explored.

The message here is: yes, there are limitations — but they're being tackled rapidly. We're not waiting for a quantum miracle. We're building toward it, step by step.



Let's now look toward the future. What can we expect in the next few years in this field?

Hardware Improvements:

Companies like IBM, Google, and Rigetti are working on increasing qubit count and fidelity.

As we move toward error-corrected qubits, deeper and more complex circuits will become feasible .

This will open the door for quantum models that rival or outperform classical deep networks.

Quantum-native Architectures:

We're already seeing research into quantum LSTMs, quantum GANs, and quantum convolutional networks.

These are not classical architectures ported to quantum — they're new structures that leverage entanglement and superposition in unique ways.

Improved Data Encoding:

Encoding remains a major bottleneck. Work is being done on more efficient encoders like qubit-efficient Fourier encodings or compressed amplitude encodings.

Hybrid Pipelines:

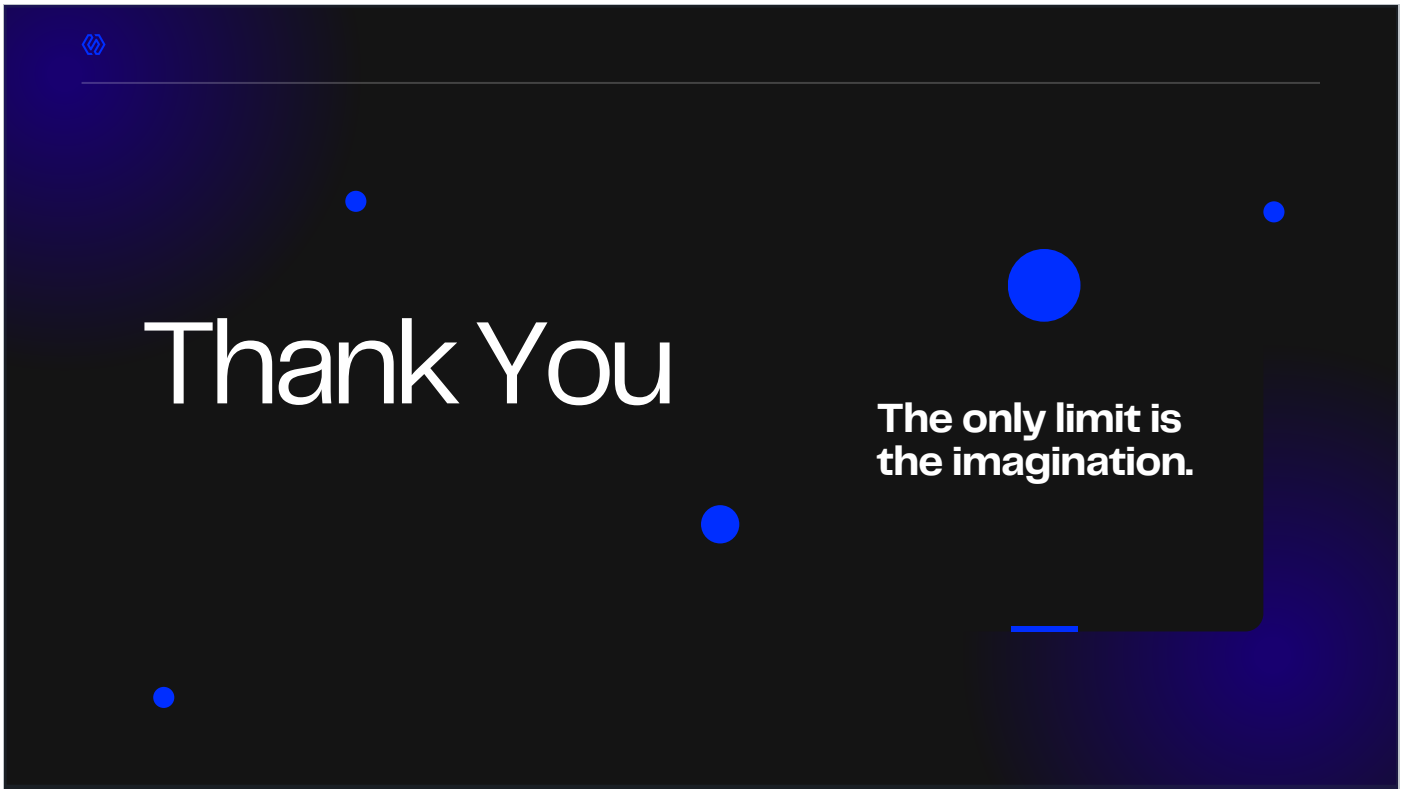
In real-world deployments, we might use classical models for preprocessing — for example, dimensionality reduction or denoising — and then pass this refined data to a quantum sub-module for prediction.

These hybrid pipelines could become standard in forecasting engines.

Use Case Expansion:

Beyond finance and IoT, quantum forecasting might play a role in climate modeling, epidemiology, and neurological signal prediction — any field where complex temporal dynamics exist.

In short, the future of quantum forecasting is about building more robust models, running them on more powerful hardware, and solving harder, more real-world problems.



Thank you so much for your attention.

To wrap up:

We've looked at the fundamentals of time series forecasting.

We explored classical and quantum models.

We built a quantum regressor using Qiskit.

And we saw how it performs — both visually and numerically.

While quantum forecasting is still in its early days, it's clear that it offers new ways to tackle old problems. It may not yet outperform classical deep learning across the board — but it's already viable in specific cases. And as hardware matures, its potential will only grow.

This talk wouldn't have been possible without the guidance of Dr. Trivedi Harsh Chandrakant, and the support from the LNMIIT community. I'm deeply grateful for that.

If you're interested in exploring further, I encourage you to install Qiskit, play with the notebooks, and start experimenting. The best way to understand quantum computing is to get your hands on it .

I’ll now take questions — or if you’d like to connect afterward, feel free to reach out. Thank you again!