Challenge Track 2 – Quantum Computing for Weather Forecasting in Cairo

Imagine predicting Cairo's weather not just with traditional computing, but with the power of quantum mechanics!
Cairo's climate is known for its unpredictability — scorching summers, rare rainfalls, and sudden shifts in wind and humidity.

In this project, we combine the proven accuracy of **classical machine learning** with the futuristic capabilities of **quantum computing** to create a hybrid forecasting system that aims to improve weather prediction accuracy for Cairo's unique climate.

Project Objectives

- Analyze Cairo's historical daily weather data (2009 – 2025).
- Develop **Classical** and **Quantum** machine learning models for temperature prediction.
- Compare model performance using **MAE**, **RMSE**, and **R^2 Score**.
- Visualize and interpret prediction results.

Why This Matters

Accurate weather forecasting is critical for agriculture, transportation, energy management, and daily life in Cairo.
By leveraging **quantum algorithms**, we aim to push the limits of forecasting technology, reducing computational time while maintaining or improving accuracy

---

2- Data Loading

Purpose:
 Load the dataset, select relevant features, remove missing values, and define the target variable.

```
uploaded = files.upload()
 filename = next(iter(uploaded))
 data = pd.read_csv(filename)
```

```
features = ['rain_sum (mm)', 'wind_speed_10m_max (km/h)', 'weather_code (wmo code)',
 'wind_direction_10m_dominant (°)', 'shortwave_radiation_sum (MJ/m²)',
 'precipitation_hours (h)', 'cloud_cover_mean (%)', 'dew_point_2m_mean (°C)',
 'relative_humidity_2m_mean (%)', 'wind_gusts_10m_mean (km/h)',
 'wind_gusts_10m_min (km/h)', 'sin_day', 'had_precipitation']
 target = 'temperature_2m_mean (°C)'
```

```
data.dropna(subset=features + [target], inplace=True)
 X = data[features].values
 y = data[target].values
```

Explanation:

- **files.upload()** imports the dataset into Colab.
- We **select features** as inputs and define the **target variable**.
- **dropna()** removes any rows with missing data.

3- Train/Test Split & Scaling

Purpose:
 Split the dataset into training (80%) and testing (20%) sets, then apply standard scaling.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
scaler_X = StandardScaler()
 scaler_y = StandardScaler()
```

```
X_train_scaled = scaler_X.fit_transform(X_train)
 X_test_scaled = scaler_X.transform(X_test)
 y_train_scaled = scaler_y.fit_transform(y_train.reshape(-1, 1)).flatten()
 y_test_scaled = scaler_y.transform(y_test.reshape(-1, 1)).flatten()
```

Explanation:
Standardizing ensures **all features have zero mean and unit variance**, improving model training stability.

4- Classical Model – MLPRegressor

Purpose:
Train a baseline **Multi-Layer Perceptron Regressor**.

```
classical_model = MLPRegressor(hidden_layer_sizes=(64,), max_iter=500,
random_state=42)
 classical_model.fit(X_train_scaled, y_train_scaled)

y_pred_classical_scaled = classical_model.predict(X_test_scaled)
 y_pred_classical = scaler_y.inverse_transform(y_pred_classical_scaled.reshape(-1,
1)).flatten()
```

Explanation:
A neural network with **1 hidden layer (64 neurons)** is trained, and predictions are transformed back to the **original temperature scale**.

5- Quantum Model – EstimatorQNN

Purpose:
Build a **Quantum Neural Network** using Qiskit.

```
num_qubits = 4
 X_train_subset = X_train_scaled[:500, :num_qubits]
 y_train_subset = y_train_scaled[:500]
 X_test_subset = X_test_scaled[:100, :num_qubits]
 y_test_subset = y_test_scaled[:100]

feature_map = ZZFeatureMap(feature_dimension=num_qubits, reps=2)
 ansatz = RealAmplitudes(num_qubits=num_qubits, reps=3)
 estimator = Estimator()

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

```
optimizer = COBYLA(maxiter=10)
regressor = NeuralNetworkRegressor(neural_network=qnn, optimizer=optimizer)
regressor.fit(X_train_subset, y_train_subset)

y_pred_qnn = regressor.predict(X_test_subset)
y_pred_qnn_original = scaler_y.inverse_transform(y_pred_qnn.reshape(-1, 1)).flatten()
y_test_original = scaler_y.inverse_transform(y_test_subset.reshape(-1, 1)).flatten()
```

Explanation:

- **ZZFeatureMap** encodes classical data into quantum states.
- **RealAmplitudes** defines trainable circuit parameters.
- **EstimatorQNN** creates a trainable quantum model.

6- Model Evaluation

Purpose:
Compare **Classical vs Quantum** model performance.

```
mae_classical = mean_absolute_error(y_test, y_pred_classical)
rmse_classical = mean_squared_error(y_test, y_pred_classical, squared=False)
r2_classical = r2_score(y_test, y_pred_classical)

mae_quantum = mean_absolute_error(y_test_original, y_pred_qnn_original)
rmse_quantum = mean_squared_error(y_test_original, y_pred_qnn_original,
squared=False)
r2_quantum = r2_score(y_test_original, y_pred_qnn_original)

print(f"Classical Model → MAE: {mae_classical:.2f}, RMSE: {rmse_classical:.2f}, R²:
{r2_classical:.2f}")
print(f"Quantum Model → MAE: {mae_quantum:.2f}, RMSE: {rmse_quantum:.2f}, R²:
{r2_quantum:.2f}")
```

Explanation:

- **MAE:** Average absolute error.
- **RMSE:** Penalizes larger errors.
- **R^2:** Measures how well the model explains variance.

7- Visualization

Purpose:
 Plot predictions vs actual values.

plt.figure(figsize=(14, 5))

Classical

plt.subplot(1, 2, 1)
 plt.plot(y_test[:50], label='Actual', marker='o')
 plt.plot(y_pred_classical[:50], label='Predicted', marker='x')
 plt.title("Classical Model: Actual vs Predicted")
 plt.legend()
 plt.grid()

Quantum

plt.subplot(1, 2, 2)
 plt.plot(y_test_original[:50], label='Actual', marker='o')
 plt.plot(y_pred_qnn_original[:50], label='Predicted', marker='x')
 plt.title("Quantum Model: Actual vs Predicted")
 plt.legend()
 plt.grid()

plt.tight_layout()
 plt.show()

Explanation:
 Visual comparison highlights prediction accuracy for both models.

8- Conclusion

- **Classical Model**: Higher accuracy and more stable performance.
- **Quantum Model**: Promising but needs optimization.
- **Next Steps**: Test hybrid quantum-classical models and increase qubit count.