Hosted by African Quantum Consortium

Data-Driven Quantum Error Mitigation (QEM)

Hackathon
Submission

 # Team 10

Maha Metawei[1,*] , Amal Bouazza[2], Jibril Abdullahi[3]

[1] Doctor of Philosophy, Researcher at Electronics Research Institute, Cairo, Egypt
[2] PhD Student at CNRS-CRAN, University of Lorraine, France and The International University of Rabat, Morrocco
[3] B.Sc. Pure and Applied Physics, Aliko Dangote University; currently with Quantum Africa (R&D)
[*] Leader of the team

Where this section fits in the full hackathon deck

1. **Problem Statement**
   Why QEM & what we predict ?
2. **Dataset generation**
   Circuit sweep & ideal/noisy features
3. **Training**
   Model setup & hyperparameters
4. **Results**
   Accuracy vs noise strength & ablations
5. **Demo / takeaways**
   How to use it ?

End-to-end generation pipeline

Why QEM?

- Real quantum hardware introduces noise measured expectation values are biased.
- Many quantum workflows (variational algorithms, optimization, control) rely on accurate $< O >$.
- Goal : **Recover the ideal (noise-free) expectation** value from noisy measurements.

Our task:

- **Inputs** : noisy expectation values under multiple noise settings.
- **Target** : ideal expectation value computed noise-free.

$$\hat{x}_i = f\left(\left\{x_n^{(type,\,rate)}\right\}, num_{qubits}, depth\right)$$

Deliverable:

- A model that outputs a **mitigated estimate** $\hat{x}_i$ usable directly in the rest of the pipeline.

Where this section fits in the full hackathon deck

1 **Problem Statement**
Why QEM & what we predict ?

2 **Dataset generation**
Circuit sweep & ideal/noisy features

3 **Training**
Model setup & hyperparameters

4 **Results**
Accuracy vs noise strength & ablations

5 **Demo / takeaways**
How to use it ?

End-to-end generation pipeline

Circuit sweep
qubits ∈ ⟦2,5⟧
layers ∈ ⟦1,10⟧

→

Random params
θ ~ Uniform(0, 2π)
200 circuits / config

→

Ideal label
Statevector
⟨O⟩ (noise-free)

→

Noisy features
3 noise types × 6 rates
AzeSimulator (shots= 2048)

→

CSV export
8000 rows
23 columns total

The output is a supervised dataset for QEM

We cast QEM as supervised learning: predict the noise-free expectation from noisy measurements.

Each row corresponds to one parameterized circuit instance (n, L, θ) evaluated on the same observable O.

The Target is $x_i = < O >$ computed with Statevector (noise-free).

The Features are 18 noisy estimates $\{x_{dep,n_0,\ldots,n_5}, x_{bit,n_0,\ldots,n_5}, x_{pha,n_0,\ldots,n_5}\}$ with 2048 shots each.

The Metadata are circuit_id, num_qubits, num_layers, num_params.

Dataset size is 8000 rows x 23 columns (1 label , 18 noisy , 4 metadata).

Learning objective : train f such that $\hat{x}_i = f(features, metadata)$

Circuit family: EfficientSU2 ansatz (linear entanglement)

Sweep configuration

Circuit generator:

$$EfficientSU2(num\_qubits, reps, entanglement = "linear")$$

- num_qubits: 2, 3, 4, 5
- reps (layers): 1 … 10
- circuits per config: 200
- parameters: $\theta \sim$ Uniform(0, 2π)

Total circuits = 4 × 10 × 200 = 8000

Why this circuit choice

- Standard variational ansatz widely used in Qiskit workflows.
- Depth is controlled by reps → easy scalability study.
- Linear entanglement matches hardware-friendly connectivity.
- Random parameter binding creates diverse circuit instances
- Good stress-test for QEM across depth and qubit count.

Key idea: same observable O; only circuit instance (n, L, θ) changes across the sweep.

Observable and ideal (noise-free) label

Nearest-neighbor ZZ correlation observable

$$O = \sum_{i=0}^{n-2} Z_i Z_{\{i+1\}}$$

Implemented as Sparse Pauli Op with terms like : ZZII … , IZZI…, ..

Label: x_i

Computed with Statevector simulation:

$x_i = < O >_i$ noise-free statevector

- No shot noise (exact expectation)
- No gate noise

→ Used as ground truth for training.

Why this label is useful ?

- Train a model to map noisy $< O >_n$ ideal expectation $< O >_i$.
- At inference: predict a **mitigated estimate** from noisy measurements.

Noisy feature generation (noise models + measurement)

Noise types × strengths

Noise families contain :

1. Depolarizing
2. Bit-flip
3. Phase-flip

Error rates: 0.01, 0.02, 0.03, 0.05, 0.08, 0.10

We set:

1Q error = p,
2Q error = 2p

Gate-level injection & measurement

Noise applied after transpilation to common gates:

- 1Q : rx, ry, rz.
- 2Q: cx

Noisy expectations $< O >$ estimated via AerSimulator (shot-based measurement):

- shots = 2048
- one estimator per (noise type, error level)

Output features: 18 noisy expectation values (3 noise types × 6 error rates).

Dataset schema (columns)

Group	Columns
Metadata (4)	circuit_id, num_qubits, num_layers, num_params
Label (1)	x_ideal
Noisy features (18)	x_dep_n0..n5, x_bit_n0..n5, x_pha_n0..n5 (n_i corresponds to error_rates[i])
Export	Saved as qem_database_large.csv (one row per circuit instance)

CSV preview (abridged)

	circuit_id	num_qubits	num_layers	num_params	x_ideal	x_dep_n0	x_dep_n1	x_dep_n2	x_dep_n3	x_dep_n4	x_dep_n5	x_bit_n0	x_bit_n1	x_bit_n2	x_bit_n3	x_bit_n4	x_bit_n5	x_pha_n0	x_pha_n1	x_pha_n2	x_pha_n3	x_pha_n4	x_pha_n5
717	715	2	4	20	0.75228018	0.5859375	0.436523438	0.366210938	0.239257813	0.104492188	0.052246094	0.554199219	0.408691406	0.337402344	0.192871094	0.087402344	0.049804688	0.571777344	0.442382813	0.345703125	0.213867188	0.091308594	0.037597656
718	716	2	4	20	0.659725337	0.502441406	0.419433594	0.327148438	0.21484375	0.099609375	0.032714844	0.484863281	0.366210938	0.278808594	0.161132813	0.042480469	0.012207031	0.533691406	0.42578125	0.354980469	0.227050781	0.138183594	0.07421875

Reproducibility & sanity checks

Reproducibility knobs

- Fixed RNG seed: np.random.seed(42)
- Deterministic sweep: qubits × layers × 200 circuits/config
- Stable observable definition per qubit count
- Consistent measurements shots: 2048

Output: qem_database_large.csv

Quick checks before training

- Confirm row count: 8000
- Confirm column count: 23
- Check ranges: noisy features should approach x_{ideal} as err → 0
- Spot-check: higher err → larger deviation (on average)

Tip: compute MAE(noisy, ideal) vs error level to validate monotonicity.

These checks ensure the training stage starts from a reliable dataset.

Where this section fits in the full hackathon deck

1 **Problem Statement**
Why QEM & what we predict ?

2 **Dataset generation**
Circuit sweep & ideal/noisy features

3 **Training**
Model setup & hyperparameters

4 **Results**
Accuracy vs noise strength & ablations

5 **Demo / takeaways**
How to use it ?

First AI Model Implementation: Random Forest Regressor

Key Objectives
- Validate and analyze quantum circuit datasets
- Explore statistical behavior of quantum noise
- Learn mappings from noisy outputs to ideal outputs
- Demonstrate AI-assisted quantum error mitigation

Dataset Description
- 8,000 quantum circuits
- 23 total features
- Target variable: x_ideal

Feature categories include circuit properties (qubits, layers, parameters) and noise channels (depolarizing, bit-flip, phase-flip).

Data Processing & Analysis
- Dataset upload and verification
- Exploratory data analysis (EDA)
- Visualization of distributions and correlations

First AI Model Implementation: Random Forest Regressor

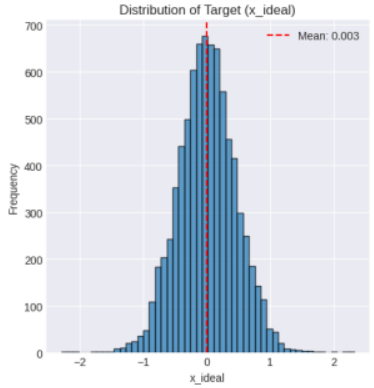
Machine Learning Model
- Random Forest Regressor
- StandardScaler preprocessing
- 80/20 train-test split

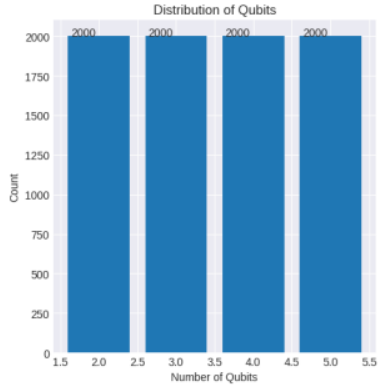
Performance
- $R^2 \approx 0.98$
- RMSE ≈ 0.062
- MAE ≈ 0.042

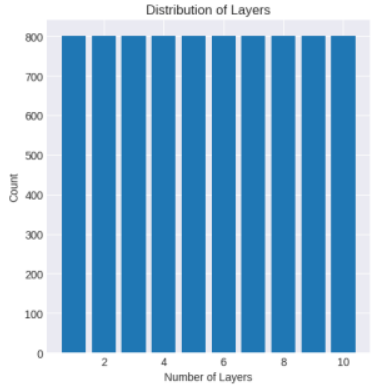
Saved Artifacts
- first_model_random_forest.pkl
- scaler.pkl

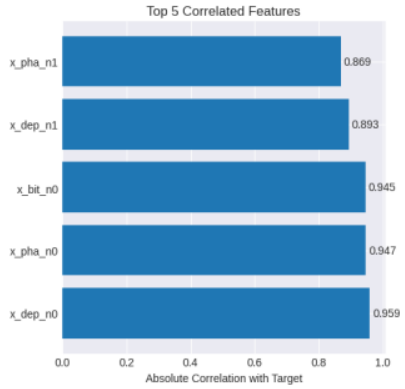
Significance
This project demonstrates how classical AI can mitigate quantum noise without additional hardware, providing a scalable solution for NISQ-era quantum devices.

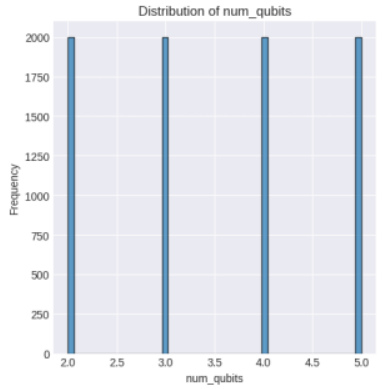
First AI Model Implementation: Results

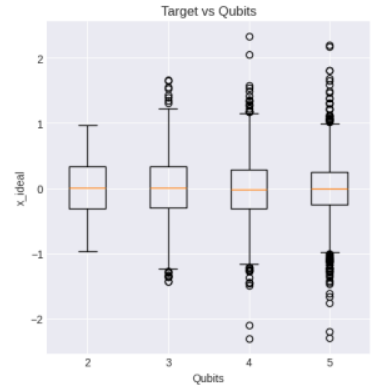


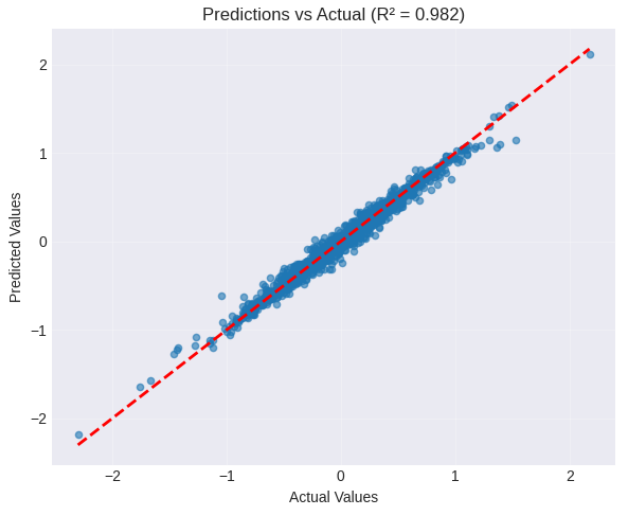


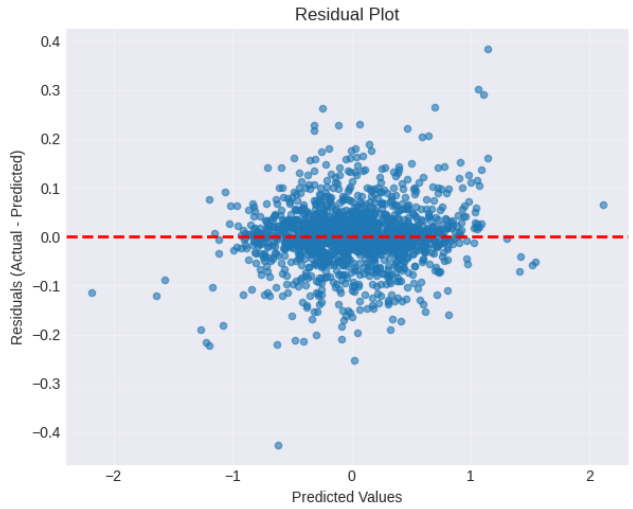












Comprehensive Model Comparison – Quantum Error Mitigation

Thirteen regression models are evaluated across different learning paradigms:

Linear Models:
- Linear Regression
- Ridge Regression
- Lasso Regression
- ElasticNet

Tree-Based Models:
- Decision Tree Regressor
- Random Forest Regressor
- Extra Trees Regressor
- Gradient Boosting Regressor

Advanced Gradient Boosting:
- XGBoost Regressor
- LightGBM Regressor

Other Models:
- k-Nearest Neighbors (k-NN)
- Support Vector Regressor (RBF)
- Multi-Layer Perceptron (Neural Network)

Where this section fits in the full hackathon deck

1 **Problem Statement**
Why QEM & what we predict ?

2 **Dataset generation**
Circuit sweep & ideal/noisy features

3 **Training**
Model setup & hyperparameters

4 **Results**
Accuracy vs noise strength & ablations

5 **Demo / takeaways**
How to use it ?

Automatic Report Generation

Final Performance Report & Model Packaging

This module generates a comprehensive, machine-readable performance report summarizing the results of all trained machine learning models for Quantum Error Mitigation (QEM). It consolidates evaluation metrics, rankings, key insights, and saved artifacts into reusable formats suitable for research, deployment, and reproducibility.

Automated Report Generation

The code constructs a structured JSON performance report (model_comparison_report.json) containing:

Project metadata and timestamp

Dataset statistics (sample count, features, train/test split)

Best-performing model and accuracy metrics

Full model ranking with evaluation scores

Summary statistics (models above R^2 thresholds)

Key scientific findings and observations

List of generated and saved model artifacts

This report enables transparent documentation and easy downstream analysis.

+ Code + Text

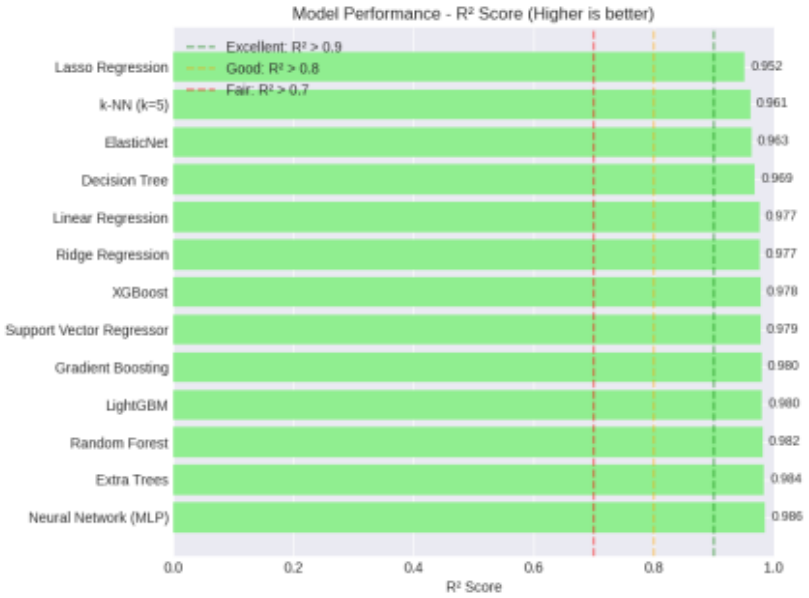
```
dnn_model.save('dnn_model.h5')
print(f"\n DNN model saved as: 'dnn_model.h5'")
```

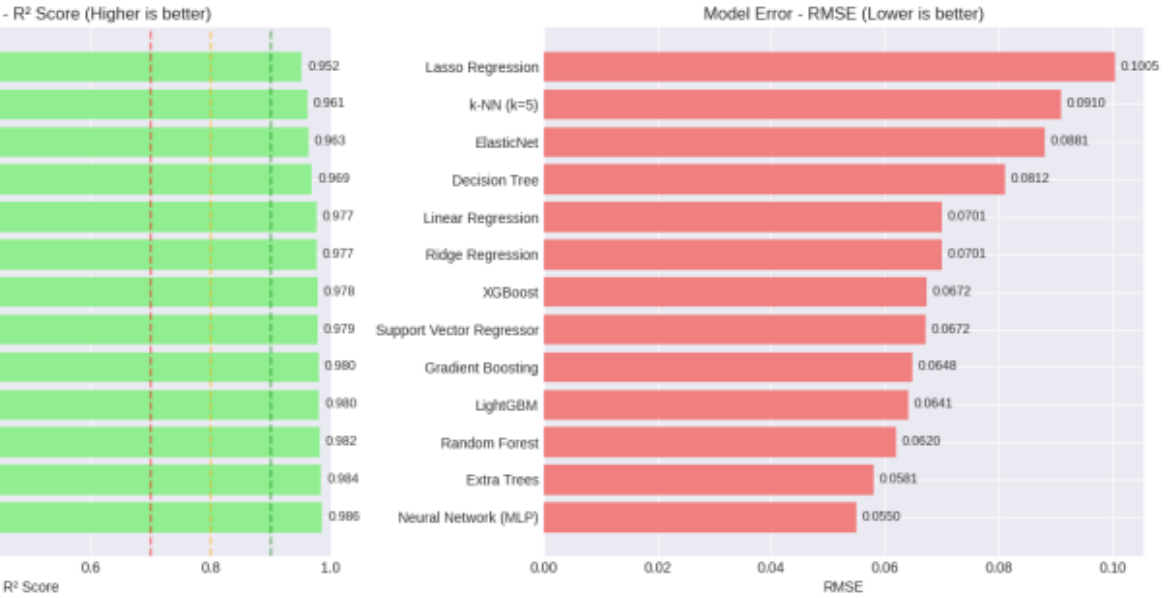
0	Neural Network (MLP)	0.985597	0.055037	0.038262	0.003029	1.834749
1	Extra Trees	0.983966	0.058070	0.038220	0.003372	2.998556
2	Random Forest	0.981743	0.061966	0.041529	0.003840	8.564224
3	Deep Neural Network	0.981149	0.062965	0.044322	0.003965	nan
4	LightGBM	0.980439	0.064141	0.041739	0.004114	0.180058
5	Gradient Boosting	0.980034	0.064801	0.044808	0.004199	3.279596
6	Support Vector Regressor	0.978524	0.067207	0.045393	0.004517	0.176362
7	XGBoost	0.978499	0.067245	0.042875	0.004522	0.339031
8	Ridge Regression	0.976662	0.070060	0.049959	0.004908	0.004197
9	Linear Regression	0.976653	0.070073	0.050019	0.004910	0.003594
10	Decision Tree	0.968681	0.081160	0.055737	0.006587	0.120090
11	ElasticNet	0.963101	0.088093	0.058036	0.007760	0.019072
12	k-NN (k=5)	0.960602	0.091028	0.061336	0.008286	0.000530
13	Lasso Regression	0.952000	0.100474	0.065965	0.010095	0.009463

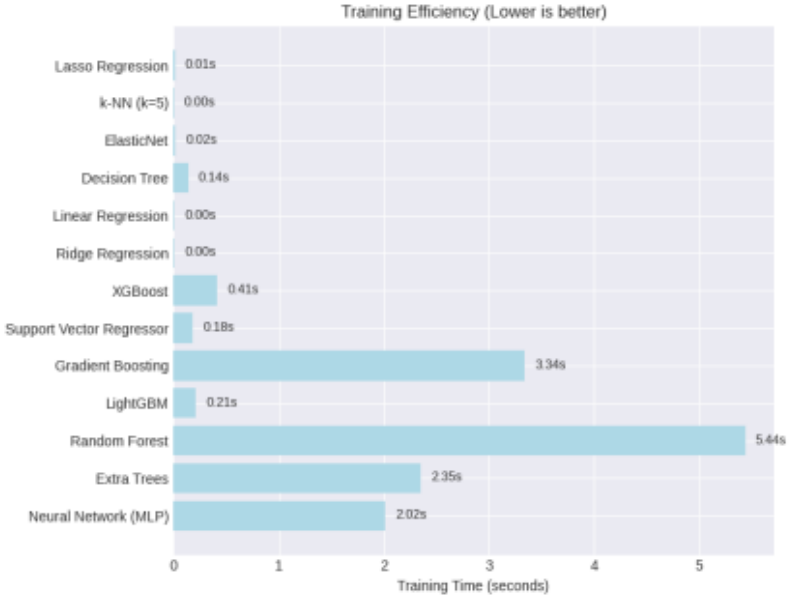
DNN Training History - Loss

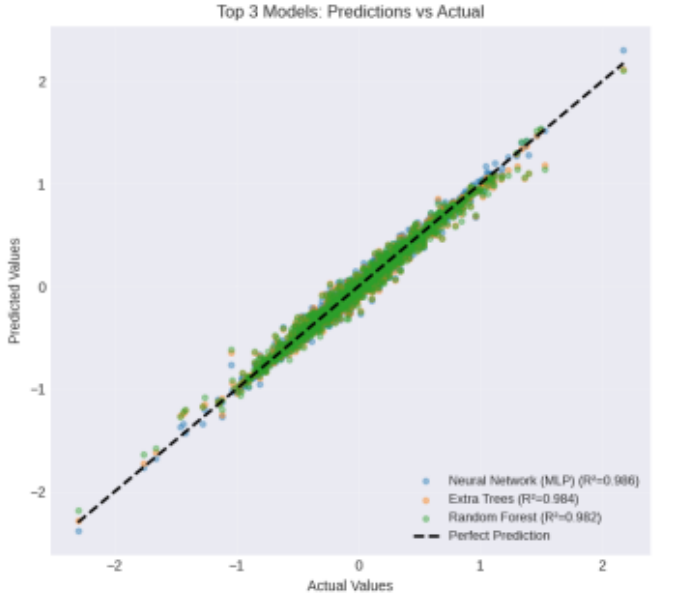
Terminal

Comprehensive Model Comparison – Results









Where this section fits in the full hackathon deck

1 **Problem Statement**
Why QEM & what we predict ?

2 **Dataset generation**
Circuit sweep & ideal/noisy features

3 **Training**
Model setup & hyperparameters

4 **Results**
Accuracy vs noise strength & ablations

5 **Demo / takeaways**
How to use it ?

Key Findings and Conclusion:

Models are ranked by R^2 score and saved to 'model_comparison_results.csv'. Visual comparisons include R^2 scores, RMSE, training time efficiency, and prediction vs actual plots for the top three models.

Key Findings

- Tree-based and ensemble models dominate performance
- Gradient boosting models (XGBoost, LightGBM) achieve top accuracy
- Linear models fail to capture nonlinear quantum noise patterns
- Neural networks show promise but require further tuning

Model Selection and Persistence

The best-performing model is automatically selected based on R^2 score and saved as 'best_model.pkl' for reuse in inference or further research.

Significance

This study demonstrates the importance of nonlinear ensemble learning techniques for data-driven Quantum Error Mitigation, particularly for NISQ-era quantum devices.

Thank You