RIDA Documentation

August 26, 2025

0 Introduction

We introduce Random Inverse Depolarizing Approximation (RIDA), which employs "random inverse" estimation circuits to estimate the error-free expectation value of an observable on noisy intermediate-scale quantum computers. We demonstrate the algorithm accurately identifies the global depolarization rate, and thereby consistently outperforms two key benchmark methods – zero-noise extrapolation (ZNE) and CNOT-only error mitigation methods – for accurate expectation value estimation across a broad range of numerical experiments on simulated quantum computers.

For ease of implementation, this documentation contains instructions to implement RIDA for two purposes:

1. To mitigate error on any quantum circuit of interest (target circuit)
2. To replicate the results of the manuscript

   > Alexander X. Miller, Micheline B. Soley, A. X. Miller, M. B. Soley, "Universal Quantum Error Mitigation via Random Inverse Depolarizing Approximation," arXiv:2508.17513 [quant-ph] (2025).

1 Instructions for General Error Mitigation

1. Import the required packages

   ```
   import numpy as np
   import qiskit
   from qiskit.dagcircuit import DAGCircuit
   from qiskit.circuit import QuantumCircuit, QuantumRegister
   ```

2. Create a Qiskit QuantumCircuit object for the circuit of interest that is both transpiled and contains measurements. Run the circuit to produce the noisy expectation value $\langle O_\epsilon \rangle$.

3. Copy the `random_inverse` and `count_gates` functions from the file `RIDA_functions.py` into the python file where RIDA will be implemented. Create a RIDA estimation circuit via the call

   ```
   random_inverse(your_circuit)
   ```

   where `your_circuit` is the name of the circuit of interest. Use the resulting circuit to produce the noisy expectation value $\langle O'_\epsilon \rangle$ via either one of the following two steps:

   (a) Run the resulting estimation circuit to directly compute $\langle O'_\epsilon \rangle$.

   (b) Repeat the call to `random_inverse` N times to generate a series of estimation circuits, run each estimation circuit independently, and average the resultant expectation values to compute a more accurate estimate of $\langle O'_\epsilon \rangle$.

4. Compute the estimate for the error-free expectation as

$$\langle O \rangle = \frac{\langle O_\epsilon \rangle}{\langle O'_\epsilon \rangle}. \tag{1}$$

   (Note that a single estimate $\langle O'_\epsilon \rangle$ is associated with a broad class of circuits similar to that associated with observable O. Thus, the same depolarization probability estimation circuit can be used for many similar quantum circuits of interest.)

2 Instructions for Reproduction of Published Results

2.1 Dependencies

Install python 3.10 with the packages

- matplotlib 3.8.3
- numpy 1.26.4
- scipy 1.12.2

- qiskit 1.0.2
- qiskit-aer 0.14.0.1
- qiskit-algorithms 0.3.0
- qiskit-ibm-runtime 0.22.0

or pull an image with the packages from dockerhub

```
docker pull alexandermiller019/rida:primary
```

Note for high-throughput computing (HTC) tests, HTCondor is required; however, individual tests require only a local machine.

2.2 Test Creation and Execution

2.2.1 On High-Throughput Computing Resources

1. To run all default tests, call

   ```
   condor_submit RIDA.sub
   ```

2. To create new test cases, call

   ```
   condor_submit Expectations.sub
   ```

   (Execution of each set of submitted tests completes within approximately half a day.)

WARNING: Running tests on HTC will cause output files to overwrite existing files with the same name. To avoid this, move existing `RESULTS` files before running new test cases, and move existing `paulis` and `params` files before generating new test cases.

2.2.2 On a Local Machine

1. To run a single test, execute `RIDA.py` for each qubit number and error multiplier.

   (Execution of all qubit number and error multiplier combinations considered, sequentially and continuously, requires approximately three days. This execution time can be reduced by decreasing the total number of tests `max_strings` in `RIDA.py`.)

2. To create a new test case, manually adjust and run `Expectations.py`.

   (Execution requires approximately one day where all tests are run sequentially and continuously. Note the number of tests generated `num_strings` in `Expectations.py` can be lowered to lessen execution time.)

2.3 Visualization

To visualize the main results,

1. Generate graphs for the full panel of tests by running `Batch_Grapher.py`.
2. Generate graphs for individual tests (*i.e.*, a specific qubit number and error multiplier) by running `Individual_Grapher.py`.

2.4 Optional Tests

2.4.1 Pauli Twirling

To model coherent error and use Pauli twirling, set `twirling` and `coherent_error` to `True` in `RIDA.py`. To graph the results, add "`, twirled, coherent`" to each `file_name` in RESULTS files.

WARNING: Pauli twirling greatly increases the execution time on a classical simulator. We recommend setting `shots` in `RIDA.py` to a single number, such as `[2**20]`, or setting `max_strings` in `RIDA.py` to `50`.

2.4.2 Random Rotations in CNOT-Only Depolarization

To include random rotations in the CNOT-only depolarization method experiments, set `random_rotations` to `True` in `RIDA.py`. To graph these results, set `use_rotations` to `True` in `Individual_Grapher.py`. Note visualization of these results requires an equivalent file without random rotations.

2.4.3 Graphing Convergence of Depolarization Estimates

To graph the convergence of depolarization estimates of RIDA and CNOT-only as a function of the number of estimation circuits, run `Convergence_Tester.py` followed by `Convergence_Grapher.py`. Note these require a `RESULTS` file with test data for the corresponding number of qubits and error multiplier.

2.4.4 Graphing Theoretical Improvement of RIDA

To graph the theoretical improvement graph of RIDA vs. unmitigated, run `Theory_Grapher.py`.

Additional Information

File Descriptions

- **RIDA_functions.py :** Used for implementing RIDA on arbitrary quantum circuits (see Sec. 1).
- **RIDA.py :** Tests RIDA and other error mitigation methods and exports to `RESULTS` files.
- **Expectations.py :** Creates test cases and exports to `params` and `paulis` files.
- **Batch_Grapher.py :** Creates graphs combining all `RESULTS` files and exports to `.png` files.
- **Individual_Grapher.py :** Creates graphs for an individual `RESULTS` file and exports to `.png` files.
- **Convergence_Tester.py :** Tests convergence of RIDA and CNOT-only depolarization estimates and exports to `CONVERGENCE RESULTS` files.
- **Convergence_Grapher.py :** Graphs convergence of RIDA and CNOT-only depolarization estimates and exports to `.png` files.
- **Theory_Grapher.py :** Graphs theory of when RIDA vs. unmitigated is expected to have lower RMSE.

Additionally, test circuit information for each qubit number of interest is initially stored in a corresponding pair of pickle files labeled `params.pkl` and `paulis.pkl`, and results are stored in `RESULTS` pickle files for all error multipliers and shot numbers considered. Note if there are existing `RESULTS` files, any new results with the same error multiplier and number of qubits will have timestamps added to their file names and will not be prioritized for graphing.

Variable Descriptions

The following test case options can be adjusted in `Expectations.py`:

- **num_strings :** The number of different Pauli strings to generate.
- **nqubits :** The number of qubits used for an individual test case (local machine only).
- **base_qubits :** The smallest number of qubits used when creating multiple test cases (HTC only).
- **layers :** The number of layers to generate the test ansatz (note a change to `layers` in `Expectations.py` must be accompanied by an equivalent change to `layers` in `RIDA.py`).

The number of test cases generated by HTC can be adjusted in `Expectations.sub` by increasing the number of queued jobs. By default, each test case beyond the first will use one additional qubit.

Testing options can be adjusted in `RIDA.py`:

- **max_strings :** The maximum number of different test cases to run. If `max_strings` exceeds the number of input cases, a number of tests equal to the number of input cases will be run.
- **nqubits :** The number of qubits used for an individual test (local machine only).
- **error_multiplier :** The error multplier applied for an individual test (local machine only).
- **base_qubits :** The smallest number of qubits used when running multiple tests (HTC only).
- **nmultipliers :** The number of different error multipliers being tested (HTC only).
- **layers :** The number of layers expected on the ansatz. This must be accompanied by an equivalent change to the layers in `Expectations.py`.
- **shots :** A list of the different numbers of shots to test for the target circuits. Note these shot numbers do not apply to the error estimation benchmark circuits, which are run separately beforehand and not rerun for each test circuit. For ZNE, the number of shots is divided by three (rounded) to ensure the number of shots for the three extrapolation levels in ZNE sum to the desired total.
- **benchmark_shots :** The total number of shots to use for benchmark circuits, generated before the test circuits are run. This number applies to RIDA, CNOT-only depolarization, and TREX circuits, but not to error-scaled ZNE circuits, since these methods are specific to each test circuit.
- **benchmark_size :** The number of different circuits used to determine the depolarization probability with RIDA and CNOT-only depolarization. The `benchmark_shots` are divided evenly among each circuit.

- **pauli_sharing :** The extent to which depolarization rates are shared across different test circuits, either `full`, `partial`, or `none`. Full Pauli sharing uses one depolarization rate for all circuits, partial Pauli sharing uses the same depolarization rate for circuits that measure the same qubits, and no Pauli sharing recomputes depolarization rates for each circuit.
- **random_rotations :** Boolean for CNOT-only depolarization that represents whether to add a layer of random rotations before and after the circuit (only accurate for specific CNOT structures, such as an even number of `EfficientSU2` layers).
- **coherent_error :** Boolean that represents whether to replace the incoherent error model with coherent overrotations.
- **twirling :** Boolean that represents whether to implement Pauli twirling on two qubit gates and measurement.
- **num_twirls :** Number of different circuits to create with randomized Pauli operators (twirling only).

The number of cases tested by HTC can be changed in `RIDA.sub` by changing the number of queued jobs. In order to queue all desired jobs, the number of queued jobs should equal the desired number of qubit numbers to be tested multiplied by the desired number of error multipliers to be tested. Note that `Batch_Grapher.py` may not format properly if the number of tests is changed.

More detailed noise model information can also be adjusted in `RIDA.py`:

- **error_1q :** The probability of one qubit gate error.
- **error_2q :** The probability of two qubit gate error.
- **ro_error :** The probability of measurement error.
- **t1 :** The T1 value in nanoseconds.
- **t2 :** The T2 value in nanoseconds.
- **gate_time :** The two-qubit gate time in nanoseconds.
- **one_qubit_time_factor :** The ratio of one-qubit to two-qubit gate time.
- **coherent_rotation :** The rotation angle in radians of R_x gates applied after two-qubit gates as error (coherent error model only).

Visualization settings can be adjusted in `Individual_Grapher.py` and `Batch_Grapher.py`:

- **shot_focus :** For graphs that only use one set number of shots, the index of that number of shots within the list of possible shots.
- **file_name :** The file or list of files to graph.
- **shot_graph :** Boolean that represents whether to graph the RMSE across Pauli strings as a function of shots used, where `Batch_Grapher.py` graphs with all combinations of qubits and error multipliers.
- **prediction :** Boolean that represents whether to graph predicted shot errors for RIDA and exponential ZNE, using the same graphs as `shot_graph`, but without CNOT-only depolarization.
- **paulis :** Boolean that represents whether to graph scatter plot with error as a function of error-free expectation of each Pauli string, where `Batch_Grapher.py` graphs with all combinations of qubits and error multipliers.
- **depolarization :** Boolean that represents whether to graph unmitigated results against error-free expectation values, with a comparison to the estimated depolarization probabilities of RIDA and CNOT-only depolarization, where `Batch_Grapher.py` graphs with all combinations of qubits and error multipliers.
- **depolarization_bar :** Boolean that represents whether to graph a bar graph with the estimated depolarizations of RIDA and CNOT-only depolarization, as well as the optimal depolarization estimation that would minimize RMSE on the trial data (`Individual_Grapher` only).
- **multipliers :** Boolean that represents whether to graph the RMSE across Pauli strings as a function of the error multiplier (`Batch_Grapher` only).
- **depolarization_summary :** Boolean that represents whether to graph RIDA, CNOT-only depolarization, and optimal estimated depolarization error strengths as a function of error multiplier (`Batch_Grapher` only).
- **use_rotations :** Boolean that represents whether to include CNOT-only depolarization with random rotations as a test case in addition to CNOT-only depolarization without random rotations (`Individual_Grapher` only, `shot_graph` and `depolarization_summary` only). Files with and without random rotations must be accessible, and the file name listed should be the version without random rotations. In order to graph exclusively with random rotations, instead change file names to match the files with rotations.

- **truncate :** Boolean that represents whether to truncate points that exceed an error of 1.1 (`Batch_Grapher` only, `paulis` only)

The following settings were used to format graphs in published results:

- **compress :** Boolean that represents whether to use a smaller subset of graphs for graphs with multiple subplots, and also save legends for these graphs to a separate file (`Batch_Grapher` only, `shot_graph` and `multipliers` only).
- **separate_plegend** Boolean that represents whether to save legends on Pauli graphs to a separate file (`Individual_Grapher` only, `paulis` only)

Bounds on the theoretical RIDA vs. unmitigated graphs can be adjusted in `Theory_Grapher.py`:

- **shot_bounds :** Lower and upper bounds on the number of shots to include in the graph (y-axis)
- **depo_bounds :** Lower and upper bounds on the depolarization probabilities to include in the graph (x-axis). This should not exceed `[0,1]`.
- **npoints :** Number of points to sample along the x-axis when plotting the cutoff of RIDA vs. unmitigated

Settings for testing the convergence of the depolarization estimates can be adjusted in `Convergence_Tester.py`:

- **nqubits :** The number of qubits used for testing.
- **error_ multiplier :** The error multiplier used for testing.
- **layers :** The number of layers expected on the ansatz. This must be accompanied by equivalent changes to the layers in `Expectations.py` and `RIDA.py`.
- **shots_per_circuit :** The number of shots per depolarization estimation circuit
- **max_circuits :** The largest number of estimation circuits simulated
- **pool_size :** The number of total estimation circuits to add to a pool of estimated depolarization values
- **nsubsets :** How many randomized subsets of pooled depolarization estimates to use for each different number of circuits

Then the following can be adjusted in `Convergence_Grapher.py`:

- **file_name :** The name of the `CONVERGENCE RESULTS` file to graph

The colors can also be adjusted for each graph:

- **raw_color :** The color used to signify the unmitigated results.
- **zne_color :** The color used to signify the results from exponential ZNE plus TREX.
- **cnot_color :** The color used to signify the results from CNOT-only depolarization plus quadratic ZNE.
- **rida_color :** The color used to signify the results from RIDA.

The following file system variables can also be adjusted in each `.py` file:

- **save_results :** Boolean that represents whether to save the results of the `.py` file.
- **print_results :** Boolean that represents whether to display results directly in output.
- **progress_flags :** Boolean that represents whether to display stage of computation and estimated time remaining (`RIDA.py` and `Expectations.py` only, local machine only).