




LogicalQ: A toolkit for quantum circuit development with built-in, generalized quantum error mitigation, detection, and correction

Rasmit Devkota^{1,2}, Ben Hagan³, Nolan Heffner^{1,2}, Younos Hashem¹, Noah Song², Arhan Deshpande¹, Fiyin Makinde⁴, Richard Yu^{2,5}, Alisa Petrusinskaia⁵, Zhixin Song¹, David Lloyd George⁶, and Lance Lampert⁶

¹ School of Physics, College of Sciences, Georgia Institute of Technology, Atlanta, GA ² School of Mathematics, College of Science, Georgia Institute of Technology, Atlanta, GA ³ Ming Hsieh Department of Electrical and Computer Engineering, Viterbi School of Engineering, University of Southern California, Los Angeles, CA ⁴ School of Electrical and Computer Engineering, College of Engineering, Georgia Institute of Technology, Atlanta, GA ⁵ School of Computer Science, College of Computing, Georgia Institute of Technology, Atlanta, GA ⁶ Department of Physics, Duke University, Durham, NC

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Open Journals](#) 

Reviewers:

- [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](https://creativecommons.org/licenses/by/4.0/)).

Summary

LogicalQ is a Python toolkit for quantum circuit development with built-in, generalized quantum error mitigation, detection, and correction (QEMDAC). LogicalQ inherits many of its data structures from, and thus is designed to interface well with, the Qiskit and Qiskit Aer packages ([Javadi-Abhari et al., 2024](#)).

The source code for LogicalQ is available on GitHub at <https://github.com/GT-Quantum-Computing-Association/LogicalQ/>. It can be installed via pip from the pypi index at <https://pypi.org/project/LogicalQ/>. Its documentation is hosted publicly at <https://logicalq.readthedocs.io/>.

Statement of need

Quantum computing presents a new model for computation which may significantly accelerate discovery in many fields, from physics ([Feynman, 1982](#)) to cryptography ([P. W. Shor, 1994](#)) to economics ([Herman et al., 2023](#)). However, the current era of quantum hardware is still noisy, and quantum error mitigation (QEM) ([Viola & Lloyd, 1998](#)), quantum error detection (QED) [[Leung1999](#)], and quantum error correction (QEC) ([Peter W. Shor, 1995](#)) techniques are necessary for the execution of utility-scale algorithms with any reasonable fidelity. Although there exist a number of libraries which allow researchers with some background to run experiments focused on QEC, there is no unified framework for general QEMDAC which is accessible to quantum computing researchers of any background and simultaneously supports the convenient implementation of quantum algorithms.

Many of the necessary components for QEMDAC have been formalized mathematically such that algorithms can be designed to construct these components for general classes of error control techniques ([Gottesman, 1997](#)). LogicalQ, like many existing QEMDAC libraries, uses such generalized constructions to meet any use case and application.

A comparison of existing libraries is made in Table 1.

Feature	LogicalQ	stim	mqt-qecc	PECOS	stac
Stabilizer code QEC	✓	✓	✓	✓	✓
Arbitrary Clifford gates	✓	✓	×	✓	✓
Arbitrary non-Clifford gates	✓	×	×	✓	✓
Fault-tolerant gates	✓	×	×	\$ check-mark\$	×
Two-way interoperability	✓	✓	×	×	✓
General noise model support	✓	×	×	✓	×
Cloud hardware interfaces	✓	×	×	×	×
Experiment suite	✓	×	×	×	×
Optimized QEC cycle scheduling	✓	×	×	×	×

Table 1: Comparison of LogicalQ with other major QEMDAC packages

In summary, many of the existing libraries are notable for their high-performance simulations and advanced implementations of certain features, but none support the full functionality required for QEC DAM applied to quantum algorithms research, especially on cloud hardware. LogicalQ is also unique in that it has a suite of experiments for testing QEC DAM which serves as a quick set of tests for researchers studying noise control.

LogicalQ was designed to accelerate the application of QEMDAC in quantum algorithm development, so its core design principle is maximizing user capability for implementing complex quantum circuits and using QEMDAC. The combination of generalized quantum error correction functionality, compatibility with libraries such as Qiskit, existence of numerous demo notebooks, and overall usability will increase accessibility to quantum error corrected-research and enable deeper study into the application of quantum error correction.

Furthermore, QEMDAC techniques can make analysis of quantum computation results difficult because they utilize overhead resources which exponentially increase the size of experiment outputs. There is a need for tools which can parse QEMDAC results without requiring researchers to understand the often-complex mathematics of these techniques.

Although many of the necessary tools are not particularly lengthy or convoluted in their implementation, LogicalQ provides a single toolkit which handles the complexities of the QEMDAC workflow to avoid user error when constructing circuit components or performing mathematical analyses.

Functionality

LogicalQ consists of various modules which provide functionality for the construction of QEMDAC components as well as their application and analysis.

A general flowchart of library usage is shown in Figure 1.

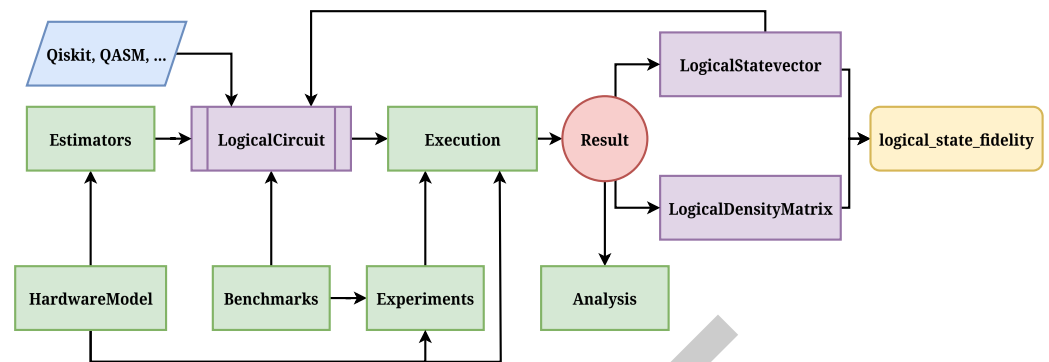


Figure 1: LogicalQ Architecture

The Logical module lies at the heart of the library with the LogicalCircuit class, which inherits from the QuantumCircuit class in Qiskit and extends it with a variety of QEMDAC features. A LogicalCircuit can be constructed from a Qiskit QuantumCircuit via the from_physical_circuit method, which enables easy integration of LogicalQ into existing workflows. The optimize_qec_cycle_indices method of LogicalCircuit performs cost accounting based on a user-provided constraint model and effective threshold in order to construct an optimal list of QEC cycle indices.

The Logical module also contains the LogicalStatevector and LogicalDensityMatrix classes, which inherit from the Statevector and DensityMatrix classes in Qiskit respectively and enable representation and analysis of quantum states at either the logical level or physical level. Logical also contains the logical_state_fidelity function which is designed to support complex fidelity comparisons, such as the fidelity of a physical density matrix and a logical statevector.

The Benchmarks module contains constructors for many of the most commonly-used benchmarking circuits in quantum computation, including randomized benchmarking and quantum volume. These functions expose parameters such as qubit counts, circuit lengths, and random selection seeds to the user so that they can be directly integrated into controlled tests and experiments.

The Experiments module contains a variety of experiments which can be used to study QEMDAC techniques. Experiment data can be analyzed with functions from the Analysis module.

The Execution module contains the execute_circuits method, which provides a single interface for both simulator and hardware backends with smart handling of complex aspects such as backend communication, hardware models, and transpilation.

The Estimators module contains special experiments which are used in QEC cycle scheduling. In particular, this includes effective threshold estimation and constraint model construction.

The Library modules contain utilities such as quantum codes for QED and QEC, hardware models for modelling quantum devices, special gates for benchmarking, and dynamical decoupling sequences for QEM.

Scholarly Work

LogicalQ development has largely been driven by an ongoing research project to optimize the scheduling of QEMDAC components in quantum circuits, with the motivation of performing fault-tolerant Hamiltonian simulations of lattice gauge theories and other physical models on quantum hardware. This involves code switching between QEC and QED codes depending on

the error-criticality of a part of a circuit, made less complex by LogicalQ's generalized framework for stabilizer codes. There is also ongoing work on genetic algorithm-based optimization of physical and logical dynamical decoupling sequences for these applications and others in science.

Acknowledgements

We acknowledge contributions from past and present members of the Quantum Computing Association at Georgia Tech.

We would like to thank Jeff Young for serving as the advisor of the Quantum Computing Association at Georgia Tech and this project.

References

- Feynman, R. P. (1982). Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6–7), 467–488. <https://doi.org/10.1007/bf02650179>
- Gottesman, D. (1997). *Stabilizer codes and quantum error correction*. arXiv. <https://doi.org/10.48550/ARXIV.QUANT-PH/9705052>
- Herman, D., Googin, C., Liu, X., Sun, Y., Galda, A., Safro, I., Pistoia, M., & Alexeev, Y. (2023). Quantum computing for finance. *Nature Reviews Physics*, 5(8), 450–465. <https://doi.org/10.1038/s42254-023-00603-1>
- Javadi-Abhari, A., Treinish, M., Krsulich, K., Wood, C. J., Lishman, J., Gacon, J., Martiel, S., Nation, P. D., Bishop, L. S., Cross, A. W., Johnson, B. R., & Gambetta, J. M. (2024). *Quantum computing with Qiskit*. <https://doi.org/10.48550/arXiv.2405.08810>
- Shor, P. W. (1994). Algorithms for quantum computation: Discrete logarithms and factoring. *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 124–134. <https://doi.org/10.1109/SFCS.1994.365700>
- Shor, Peter W. (1995). Scheme for reducing decoherence in quantum computer memory. *Phys. Rev. A*, 52, R2493–R2496. <https://doi.org/10.1103/PhysRevA.52.R2493>
- Viola, L., & Lloyd, S. (1998). Dynamical suppression of decoherence in two-state quantum systems. *Phys. Rev. A*, 58, 2733–2744. <https://doi.org/10.1103/PhysRevA.58.2733>