

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

Rasmit Devkota<sup>1,2</sup>, Ben Hagan<sup>3</sup>, Noah Song<sup>2</sup>, Zhixin Song<sup>1</sup>, David Lloyd George<sup>4</sup>, Younos Hashem<sup>1</sup>, Nolan Heffner<sup>1,2</sup>, Fiyin Makinde<sup>5</sup>, Alisa Petrusinskaia<sup>6</sup>, Richard Yu<sup>2,6</sup>, and Arhan Deshpande<sup>1</sup>

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

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Published: unpublished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](#)).

## 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 to cryptography to economics. However, the current era of quantum hardware is still noisy, and quantum error mitigation (QEM), quantum error detection (QED), and quantum error correction (QEC) techniques are necessary for the execution of utility-scale algorithms with any reasonable fidelity. Moreover, the broader scientific consensus is that these techniques will always be necessary to some extent. Although there exist a number of libraries which allow researchers with some background to run experiments focused on QEC, there is a lack of a unified framework for general QEMDAC which is accessible to quantum computing researchers of any background and simultaneously supports diverse 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. LogicalQ, like many existing QEMDAC libraries, makes use of these generalized constructions in order to support arbitrary techniques.

The Stim library is notable for its high-performance stabilizer-based simulations involving Pauli operators, but this limits its applicability to utility-algorithms which require arbitrary Clifford and non-Clifford operators which the library does not support. In contrast, LogicalQ supports multiple basis gate sets for universal quantum computation.

The mqt-qecc library, part of the Munich Quantum Toolkit, is similar to LogicalQ in its interoperability with other libraries such as Stim and Qiskit, but its functionality focuses on

41 state preparation for CSS codes and decoding for various classes of codes. However, it lacks  
42 sufficient functionality for algorithm development and experiment design.

43 The PECOS library is closer to LogicalQ in that it features support for a complete QEC protocol  
44 for general stabilizer codes and more general noise models, but it is limited in native gate  
45 support and its lack of built-in interoperability with standard libraries for quantum circuit  
46 development introduces friction in the application of QEC to quantum algorithms research.

47 The stac library also has many of the above features but also supports non-stabilizer circuit  
48 simulations with non-Clifford operators, such as arbitrary rotations. Due to the Gottesman-Knill  
49 theorem, stabilizer circuits cannot produce quantum advantage over classical computers, so  
50 this functionality is necessary for advances in quantum algorithm research.

Feature	LogicalQ	mqt-qecc	PECOS	stac	stim
Stabilizer code QEC	✓	✓	✓	✓	✓
Arb. Clifford Ops	✓	×	✓	×	✓
Arb. Non-Clifford Ops		×	×	×	×
Optimized QEC Cycle Scheduling	✓	×	×	×	×
Two-way Qiskit transpilation	✓	×	×		✓
QASM export	×	×	×	✓	✓
Encoding of custom circuits	✓	×		×	
FT gate implementation	✓	×		×	×

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

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

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

## 65 Functionality

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

68 The Logical module lies at the heart of the library with the implementation of the  
69 LogicalCircuit class. This class inherits from the QuantumCircuit class in Qiskit and  
70 extends it with a QEMDAC feature set. This module also contains the implementations  
71 of the LogicalStatevector and LogicalDensityMatrix classes (which inherit from the  
72 Statevector and DensityMatrix classes in Qiskit, respectively), which enable direct  
73 representation and analysis of quantum states at either the logical level or physical level.  
74 Logical also contains the logical\_state\_fidelity function which is designed to support  
75 complex fidelity comparisons, such as the fidelity of a physical density matrix and a logical  
76 statevector.

77 An important feature of the LogicalCircuit class is its `from_physical_circuit` class method,  
78 which allows interoperability with Qiskit (or, in fact, any tool which can export OpenQASM code  
79 which can then be imported into a QuantumCircuit). Because of the inheritance structure,  
80 most of the familiar class attributes and methods are available LogicalCircuit, including  
81 many which have been overridden with special behavior for QEMDAC. This includes logical  
82 realizations of common quantum gates, many of which can be realized using different methods.

83 The `optimize_qec_cycle_indices` method of LogicalCircuit performs cost accounting  
84 based on a user-provided constraint model and effective threshold in order to construct an  
85 optimal list of QEC cycle indices for each logical qubit.

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

91 The Experiments module contains a variety of experiments which can be used to study  
92 QEMDAC techniques. It also contains the `execute_circuits` method, which provides a unified  
93 interface with both simulator and hardware backends. Experiment data can be analyzed with  
94 functions from the Analysis module, as long as they conform to the expected data structures for  
95 each analysis function. The NoiseModel module and the Library.HardwareModels submodule  
96 provide multiple utilities for injecting noise of known parameters into experiments.

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

## 99 Scholarly Work

100 LogicalQ development has largely been driven by an ongoing research project to optimize the  
101 scheduling of QEMDAC components in quantum circuits, with the motivation of performing  
102 fault-tolerant Hamiltonian simulations of physical theories on quantum hardware.

## 103 Acknowledgements

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

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

## 108 References

109 Javadi-Abhari, A., Treinish, M., Krsulich, K., Wood, C. J., Lishman, J., Gacon, J., Martiel, S.,  
110 Nation, P. D., Bishop, L. S., Cross, A. W., Johnson, B. R., & Gambetta, J. M. (2024).  
111 *Quantum computing with Qiskit*. <https://doi.org/10.48550/arXiv.2405.08810>