The Journal of Open Source Software

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

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

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** Department of Physics, Duke University, Durham, NC **5** School of Electrical and Computer Engineering, College of Engineering, Georgia Institute of Technology, Atlanta, GA **6** School of Computer Science, College of Computing, Georgia Institute of Technology, Atlanta, GA

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

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

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

The `stac` library also has many of the above features but also supports non-stabilizer circuit simulations with non-Clifford operators, such as arbitrary rotations. Due to the Gottesman-Knill theorem, stabilizer circuits cannot produce quantum advantage over classical computers, so 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	✓	×		×	×

`LogicalQ` was designed to accelerate the application of QEMDAC in quantum algorithm development. Thus, 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.

The `Logical` module lies at the heart of the library with the implementation of the `LogicalCircuit` class. This class inherits from the `QuantumCircuit` class in `Qiskit` and extends it with a QEMDAC feature set. This module also contains the implementations of the `LogicalStatevector` and `LogicalDensityMatrix` classes (which inherit from the `Statevector` and `DensityMatrix` classes in `Qiskit`, respectively), which enable direct 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.

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

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 for each logical qubit.

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

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

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

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