An MLIR Dialect for Fault-Tolerant Quantum Computing

Narasinga Rao Miniskar*
Oak Ridge National Laboratory
Oak Ridge, TN 37830, USA
miniskarnr@ornl.gov

Elaine Wong*
Oak Ridge National Laboratory
Oak Ridge, TN 37830, USA
wongey@ornl.gov

Seyong Lee*
Oak Ridge National Laboratory
Oak Ridge, TN 37830, USA
lees2@ornl.gov

Vicente Leyton-Ortega
Oak Ridge National Laboratory
Oak Ridge, TN 37830, USA
leytonorteva@ornl.gov

Jeffrey S. Vetter
Oak Ridge National Laboratory
Oak Ridge, TN 37830, USA
vetter@ornl.gov

Travis S. Humble
Oak Ridge National Laboratory
Oak Ridge, TN 37830, USA
humblets@ornl.gov

***Abstract*—To achieve fault-tolerant quantum computing (FTQC), a compilation pipeline must address quantum error-correcting codes, syndrome extraction, magic-state distillation, and physical-gate decomposition. We introduce `ftqc`, an open-source dialect for MLIR (Multi-Level Intermediate Representation) that treats these concerns as first-class constructs in the intermediate representation (IR). Logical qubits are type-safe static single-assignment (SSA) values parameterized by their error-correcting code and distance, letting the compiler enforce code and distance compatibility at the type level. The dialect provides operations covering the logical life-cycle: encoded initialization, transversal Clifford gates, T gates and Toffoli gates by magic-state injection, syndrome extraction, and distillation. A resource estimator derives physical-qubit counts and logical error rates directly from ECC metadata in the IR, and three validated backends emit OpenQASM 3, QIR (Quantum Intermediate Representation), and Stim. Across four benchmarking quantum circuits, end-to-end compilation completes in under 15 ms, and a full logical-error-rate sweep through Stim runs in under 4 s for quantum circuits of up to 52 physical qubits.**

***Index Terms*—compiler, quantum error correction, stabilizer circuits, MLIR, QIR, quantum compilation**

I. Introduction

Progress beyond noisy intermediate-scale quantum (NISQ) hardware depends on quantum error correction (QEC) [1], [2], which encodes each logical qubit in many physical qubits so that errors can be detected and corrected [3]. This encoding is expensive, since realistic algorithms are expected to need millions of physical qubits [4], [5].

Software for fault-tolerant compilation lags far behind the classical compiler stack. Qiskit [6], Cirq [7], and tket [8] target NISQ gate optimization, while intermediate representations such as PennyLane/Catalyst [9], CUDA-Q Quake [10], [11], QIR [12], and XACC [13] do not expose code-parameterized logical qubits, syndrome extraction, or magic states as first-class entities [14].

We therefore introduce `ftqc`, a domain-specific MLIR [15] dialect for fault-tolerant compilation, whose progressive lowering keeps logical and physical level in one IR. `ftqc` provides types for logical qubits, syndromes, magic states, and Pauli frames, tracking code and distance in the type: a resource estimator deriving physical-qubit counts and logical error rates from the IR, validated backends for OpenQASM 3 [16], QIR [12], and Stim [17].

*These authors contributed equally to this work and are designated as co-first authors.

II. Approach

We define the `ftqc` dialect declaratively in TableGen using MLIR's Operation Definition Specification, which generates its types, attributes, and operations. Four types model the fault-tolerant state: a logical qubit whose type records its error-correcting code and distance, for example `!ftqc.logical_qubit<#ftqc.ecc<steane,3>>`, together with syndrome-register, magic-state, and Pauli-frame types. Enumerated attributes name the code, magic, and Pauli kinds, and a `code_props` attribute carries the $[\![n, k, d]\!]$ parameters. Because this metadata is part of the type, the verifier rejects code- or distance-mismatched programs statically. The 31 operations span the logical life-cycle: initialization, transversal Clifford gates, non-Clifford T and Toffoli gates realized by magic-state injection, syndrome extraction, error correction, and magic-state distillation.

The dialect provides 18 passes operate in three groups. *Transformation* passes encode a circuit into a chosen code, insert X and Z syndrome rounds after each gate, rewrite non-transversal gates, expand each T gate into a 15-to-1 distillation factory [18], [19], and defer Pauli corrections into a frame. *Analysis* passes estimate resources from the code and distance in the type, enumerate leading-order fault paths, and check the physical error rate against the code threshold. *Lowering* passes emit executable output: OpenQASM 3 [16], QIR [12] with physical-qubit expansion, Stim stabilizer circuits [17] with DETECTOR and OBSERVABLE records for MWPM decoding [20], and IQM Resonance JSON. Composed, the passes lower a circuit from logical operations to a hardware- or simulator-ready program.

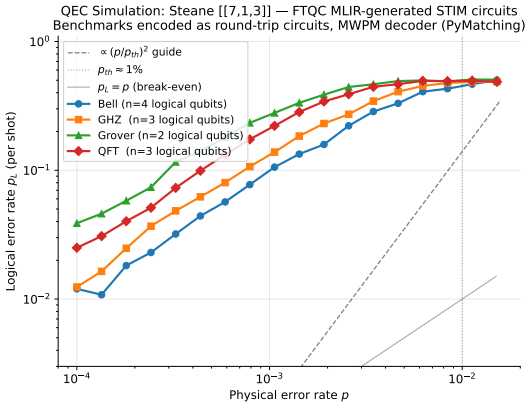


Fig. 1. FTQC+Stim per-shot circuit failure rate versus physical error rate p for the four benchmarks, Steane-encoded and decoded with MWPM (PyMatching, 5000 shots per point). The diagonal marks $p_{\mathrm{shot}}{=}p$ and the vertical line marks the code threshold $p_{\mathrm{th}} \approx 1\%$.

III. Results

We evaluated `ftqc` dialect on an Intel Core i7-12700K using LLVM/MLIR 22. The four benchmarks are Bell and GHZ-4 (Steane $d{=}3$, Clifford only), Grover-2 (surface $d{=}5$), and QFT-3 (Steane $d{=}3$, six T gates).

Magic-state distillation dominates the resource budget. From the code and distance in the type alone, the estimator reports 26 physical qubits for Bell, 52 for GHZ-4, 98 for Grover, and 234 for QFT-3. The six T gates in QFT-3 invoke a single 15-to-1 factory that alone requires 195 distillation qubits, five times the circuit's own 39-qubit footprint.

We validated correctness at three levels. The QASM3 output ran on Qiskit Aer [6] and the QIR output on QIR-EE [21], [22], where Bell and GHZ-4 gave the expected 50/50 superpositions, Grover returned $|11\rangle$ deterministically, and QFT-3 reproduced its state-vector distribution over 8192 shots, for which the per-outcome shot-sampling uncertainty $\sqrt{p(1-p)/N}$ is about 0.5 percentage points. For fault-tolerant semantics, we lowered the benchmarks to Stim as round-trip Steane-encoded circuits with three syndrome rounds per gate, decoded with PyMatching [20] over an 18-point sweep. At $p{=}10^{-3}$ the per-shot failure rates were 9.9% for Bell, 13.6% for GHZ, 27.6% for Grover, and 21.6% for QFT, and Fig. 1 plots them against p.

`ftqc`+Stim completed this sweep for all benchmarks (26 to 52 physical qubits, 5000 shots per point) in under 4 s, and every circuit compiled in under 15 ms.

The same annotations also expose space-time cost. Under parallel execution, circuit depth equals the number of EC rounds: 4, 8, 20, and 36 cycles for Bell, GHZ-4, Grover, and QFT-3. QFT-3 is bounded by its distillation factory, whose six T gates serialize through one 15-to-1 unit, so a second parallel factory roughly halves that critical path.

IV. Conclusion

We have presented `ftqc`, an open-source MLIR dialect for fault-tolerant quantum compilation. Three features distinguish it: resource estimation driven by the code and distance metadata in the IR, verified lowering to OpenQASM 3 and QIR with syndrome support, and a Stim back end for scalable Monte-Carlo QEC simulation with MWPM decoding. Across the four benchmarks, it produced accurate resource annotations in under 15 ms and scaled to 52 physical qubits in under 4 s. Because the T gates account for most of the physical-qubit overhead, reducing the T count remains the primary target for future work.

References

[1] P. W. Shor, "Scheme for reducing decoherence in quantum computer memory," *Physical Review A*, vol. 52, no. 4, pp. R2493–R2496, 1995.

[2] A. M. Steane, "Error correcting codes in quantum theory," *Physical Review Letters*, vol. 77, no. 5, pp. 793–797, 1996.

[3] D. Gottesman, "Stabilizer codes and quantum error correction," Ph.D. dissertation, California Institute of Technology, 1997.

[4] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, "Surface codes: Towards practical large-scale quantum computation," *Physical Review A*, vol. 86, no. 3, p. 032324, 2012.

[5] C. Gidney and M. Ekå, "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits," *Quantum*, vol. 5, p. 433, 2021.

[6] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross, B. R. Johnson, and J. M. Gambetta, "Quantum computing with Qiskit," https://qiskit.org, 2024.

[7] C. Developers, *Cirq: Google Quantum AI*. Zenodo, Aug. 2025. [Online]. Available: https://zenodo.org/doi/10.5281/zenodo.4062499

[8] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan, "t|ket⟩: A retargetable compiler for NISQ devices," *Quantum Science and Technology*, vol. 6, no. 1, p. 014003, 2021.

[9] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, M. S. Alam, S. Ahmed, J. M. Arrazola, C. Blank, A. Delgado, S. Jahangiri, K. McKiernan, J. J. Meyer, Z. Niu, A. Száva, and N. Killoran, "PennyLane: Automatic differentiation of hybrid quantum-classical computations," 2018.

[10] NVIDIA Corporation, "CUDA-Q: A platform for hybrid quantum-classical computing," https://github.com/NVIDIA/cuda-quantum, 2026.

[11] ——, "Quake dialect specification," https://nvidia.github.io/cuda-quantum/latest/specification/quake-dialect.html, 2026.

[12] QIR-Alliance, "Quantum intermediate representation (QIR)," https://github.com/qir-alliance/qir-spec, 2021.

[13] A. J. McCaskey, D. I. Lyakh, E. F. Dumitrescu, S. S. Powers, and T. S. Humble, "XACC: A system-level software infrastructure for heterogeneous quantum-classical computing," *IET Quantum Communication*, vol. 1, no. 1, pp. 12–24, 2020.

[14] P. Hopf, E. Ochoa, Y. Stade, D. Rovara, N. Quetschlich, I. A. Florea, J. Izaac, R. Wille, and L. Burgholzer, "Integrating quantum software tools with(in) mlir," in *Proceedings of the Supercomputing Asia and International Conference on High Performance Computing in Asia Pacific Region*, ser. SCA/HPCAsia '26. New York, NY, USA: ACM.

[15] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko, "MLIR: Scaling compiler infrastructure for domain specific computation," in *Proceedings of the International Symposium on Code Generation and Optimization (CGO)*, 2021, pp. 2–14.

[16] A. W. Cross, A. Javadi-Abhari, T. Alexander, N. de Beaudrap, L. S. Bishop, S. Heidel, C. A. Ryan, P. Sivarajah, J. Smolin, J. M. Gambetta, and B. R. Johnson, "OpenQASM 3: A broader and deeper quantum assembly language," *ACM Transactions on Quantum Computing*, vol. 3, no. 3, pp. 1–50, 2022.

[17] C. Gidney, "Stim: A fast stabilizer circuit simulator," *Quantum*, vol. 5, p. 497, 2021.

[18] S. Bravyi and A. Kitaev, "Universal quantum computation with ideal Clifford gates and noisy ancillas," *Physical Review A*, vol. 71, no. 2, p. 022316, 2005.

[19] D. Litinski, "A game of surface codes: Large-scale quantum computing with lattice surgery," *Quantum*, vol. 3, p. 128, 2019.

[20] O. Higgott, "PyMatching: A python package for decoding quantum codes with minimum-weight perfect matching," *ACM Transactions on Quantum Computing*, vol. 3, no. 3, pp. 1–16, 2022.

[21] E. Wong, V. Leyton-Ortega, D. Claudino, S. R. Johnson, A. J. Adams, S. Afrose, M. Gowrishankar, A. Cabrera, and T. S. Humble, "A cross-platform execution engine for the quantum intermediate representation," *The Journal of Supercomputing*, vol. 81, no. 16, p. 1521, 2025.

[22] QIR-EE Developers, "QIR-EE," 2025. [Online]. Available: https://github.com/ORNL-QCI/qiree