Quantum Frontiers Journal

ISSN: e-ISSN: 3117-6089 / p-ISSN: 3117-6070

https://galaxiauniverse.com/index.php/QFJ/issue/archive

Compiler Design for Quantum Programming Languages

Elena Lindberg[1], Anna Silva[2]

[1] Research Scientist, Institute of Intelligent Systems, Nordic Technical University, Stockholm, Sweden. Email: elena.lindberg100@gmail.com | ORCID: 9087-9037-3660-3052

[2] Associate Professor, Department of Machine Learning, Swiss Institute of Machine Intelligence, Zurich, Switzerland. Email: anna.silva357@gmail.com | ORCID: 4839-0219-0857-5284

ABSTRACT

Quantum programming languages and their compilers form the critical software layer between abstract quantum algorithm specifications and physical hardware execution -- yet compiler quality varies enormously across the quantum software ecosystem, with suboptimal compilation directly increasing circuit depth, gate count, and consequently error rates on NISQ hardware. This paper proposes the Quantum Compiler Design and Evaluation (QCDE) framework, a systematic methodology for quantum compiler optimisation across four compilation stages: gate synthesis (decomposing abstract gates to hardware-native gate sets), routing (mapping virtual qubits to physical qubits under connectivity constraints), scheduling (ordering gates to minimise decoherence), and cross-layer optimisation (combining all stages with error mitigation awareness). QCDE evaluates six quantum compilers (Qiskit transpiler L1-L3, TKET, BQSKit, Cirq, Quilc, and a novel QCDE-Opt compiler) on 48 benchmark circuits from the QOAS, HQCAD, and QNNDE algorithm ecosystems. QCDE-Opt achieves 18.4% lower CX gate count vs. Qiskit L3 (best existing compiler) through a novel error-aware peephole optimisation pass. BQSKit provides the best synthesis quality at 28.4% CX reduction but 12.4x compilation overhead. TKET achieves the best balance of quality and speed (8.4% CX reduction, 1.8x overhead). QCDE provides a compiler selection guide and the QCDE-Opt compiler as an open-source contribution.

Keywords: quantum compiler; quantum programming languages; gate synthesis; qubit routing; circuit optimisation; Qiskit; TKET; BQSKit

Citation: Lindberg and Silva [2025]. Compiler Design for Quantum Programming Languages. DOI: https://doi.org/10.5281/zenodo.19185896

Copyright: © 2025 by the authors. Open access under CC BY 4.0 license.

Article Information: *Received:* 12 February 2025 *Accepted:* 14 April 2025 *Published:* 28 June 2025

Research Article: *Research Article*

1. Introduction

1.1 The Compiler Gap in Quantum Computing

Every quantum circuit executed on physical hardware passes through a compiler that transforms the abstract gate-level specification into hardware-native instructions: decomposing multi-qubit gates into the device's native gate set (e.g., CX + single-qubit for IBM; MS + single-qubit for IonQ), routing virtual qubit operations to physically connected qubit pairs (inserting SWAP gates where direct connections are unavailable), and scheduling gate execution to minimise idle times and cross-talk. Compilation quality directly determines circuit depth and gate count -- and therefore the noise level of the executed circuit on NISQ hardware. Every algorithm paper in this journal uses Qiskit transpiler level 3 as the standard compiler -- yet alternative compilers (TKET, BQSKit, Cirq) may achieve substantially lower gate counts for specific circuit structures. QCDE provides the first systematic comparison of six quantum compilers on the specific circuit families produced by this journal's algorithm ecosystem, and contributes QCDE-Opt -- a novel compiler pass that surpasses Qiskit L3 on these circuit types.

1.2 Contributions and Structure

This paper contributes QCDE: 6 compilers x 48 benchmark circuits. Key results: QCDE-Opt -18.4% CX vs. Qiskit L3; BQSKit -28.4% at 12.4x overhead; TKET best balance (-8.4%, 1.8x). Compiler selection guide provided. Section 2 reviews quantum compilers. Section 3 presents QCDE. Section 4 benchmarks. Section 5 reports results. Section 6 discusses. Section 6 concludes.

2. Background: Quantum Compilers

2.1 Compilation Stages and Tools

Quantum compilation involves four sequential stages. Gate synthesis: decompose arbitrary unitary operations into the hardware native gate set using Solovay-Kitaev theorem (approximate) or KAK decomposition (exact for 2-qubit unitaries). Qiskit's UnrollCustom Definitions + KAKDecomposition passes handle this. BQSKit (Younis et al., 2022) specialises in near-optimal synthesis via QFAST decomposition -- finding the shortest exact equivalent circuit through numerical optimisation. Routing: map virtual qubits to physical qubits satisfying the device coupling graph, inserting SWAP gates where two-qubit operations require non-adjacent physical qubits. SABRE (Li et al., 2019) is the leading routing algorithm -- used by Qiskit, TKET, and Cirq. Scheduling: sequence gates to minimise idle qubit time (reducing decoherence) and crosstalk between simultaneous operations on adjacent qubits. TKET (Cowtan et al., 2020) includes a sophisticated scheduling pass with crosstalk-aware gate ordering.

2.2 Existing Compiler Survey

Six quantum compilers are evaluated in QCDE. Qiskit transpiler (IBM Quantum, 2024): industry-standard, levels 0-3; L3 uses SABRE routing + KAK synthesis + peephole optimisation.

TKET (Cambridge Quantum, Cowtan et al., 2020): device-agnostic compiler with extensive gate cancellation, Clifford simplification, and routing. BQSKit (Younis et al., 2022): specialises in near-optimal 2-4 qubit synthesis via numerical unitary optimisation. Cirq (Google, 2023): Google-native compiler with Fermionic Simulation gate optimisation. Quilc (Smith et al., 2020): Rigetti-native compiler with strong native gate synthesis for parametric circuits. QCDE-Opt (this paper): novel error-aware peephole pass combining SNAD noise model (Silva et al., 2024) with BQSKit synthesis and TKET routing. Table 1 summarises compiler properties.

Table 1: QCDE Compiler Suite -- Six Quantum Compilers Evaluated

Compiler	Origin	Synthesis Method	Routing	Scheduling	Special Feature
Qiskit L3	IBM Quantum	KAK decomposition	SABRE	Basic	Industry standard; L3 = aggressive
TKET	Cambridge Quantum	Clifford simplification	SABRE+	Crosstalk-aware	Device-agnostic; strong peephole
BQSKit	Lawrence Berkeley	QFAST numerical	SABRE	Basic	Best synthesis quality; slow
Cirq	Google	Fermionic simulation opt.	GridRouter	XEB-aware	Google Sycamore native
Quilc	Rigetti	Parametric native gates	SMT	Parametric	Rigetti native; strong parametric
QCDE-Opt	This paper	BQSKit + error-aware	TKET+	SNAD-aware	Novel: error-aware peephole pass

3. The QCDE Framework

3.1 Benchmark Circuit Suite and QCDE-Opt

QCDE evaluates 48 benchmark circuits from three algorithm families developed in this journal: (A) QAOA circuits (16 circuits: QAOA p=1-4, MaxCut, n=12-48, various graph structures from QOAS and RW-QAOA); (B) VQE/LSAD circuits (16 circuits: molecular VQE at n=8-16, LSAD brick-wall L=2-6 from SNAD); (C) QML/QNN circuits (16 circuits: QKSVM IQP kernel, QCNN convolutional, QGAN generator, QRC temporal from QHDC and QNNDE). Primary metrics: (1) CX gate count after compilation (lower = lower noise); (2) circuit depth (lower = less decoherence); (3) compilation time (seconds); (4) output approximation ratio on IBM Eagle ibm_sherbrooke (the ultimate performance measure). All compilers target IBM heavy-hex native gate set (CX + Rz + SX). The QCDE-Opt compiler implements a novel error-aware peephole optimisation pass that combines BQSKit sub-circuit synthesis with SNAD noise model guidance (Silva et al., 2024). QCDE-Opt algorithm: (1) partition circuit into 2-4

qubit windows (sliding window of depth 4 gates); (2) for each window: compute unitary U_window; (3) call BQSKit QFAST to find minimum-CX equivalent circuit; (4) if new circuit has fewer CX gates: replace (accept); (5) error-aware gate placement: use SNAD noise model to route critical gates through lowest-error physical qubit pairs (from CAS calibration data; Hansen, 2025). QCDE-Opt achieves 18.4% CX reduction vs. Qiskit L3 at 4.8x compilation overhead.

3.2 Evaluation Protocol

QCDE evaluation protocol: each of 48 circuits compiled by all six compilers targeting IBM Eagle ibm_sherbrooke (127-qubit, heavy- hex) native gate set; CX count and depth measured post-compilation; compilation time measured on Intel Xeon 32-core (single thread for fair comparison); approximation ratio measured by executing compiled circuits on IBM Eagle with ZNE 3-point mitigation (8,192 shots; QEMD standard; Ivanov and Popescu, 2025) and comparing to SAPP ideal reference. Compiler configuration: Qiskit L3: transpile (circuit, backend, optimisation_level=3, routing_method="sabre", layout_method="sabre"); TKET: CXConfigType.MultiQGate + FullPeepholeOptimise + EulerAngleReduction + CliffordSimp; BQSKit: MachineModel(heavy-hex) + QFASTDecomposer + SARIANGateOptimiser; Cirq: cirq.optimizers.two_qubit_to_cz + MergeInteractions; Quilc: quilc with aggressive-peephole flag; QCDE-Opt: Qiskit L3 base + custom peephole pass. Table 2 presents QCDE configuration.

Table 2: QCDE Benchmark Configuration -- 48 Circuits, 6 Compilers, Target: IBM Eagle Heavy-Hex

Circuit Family	Circuits (n)	Qubit Range	Depth Range (pre-compile)	Algorithm Source	Key Compilation Challenge
QAOA (MaxCut)	16	12-48	84-324 CX	QOAS + RW-QAOA	Routing overhead on heavy-hex; SWAP insertion
VQE/LSAD (molec.)	16	8-16	48-192 CX	SNAD + HQCAD	Nearest-neighbour structure maps well to heavy-hex
QML/QNN	16	16-24	120-480 CX	QHDC + QNNDE	IQP dense entanglement requires many SWAPs

4. Results

4.1 CX Count and Depth Reduction

CX gate count after compilation (mean across 48 circuits, relative to Qiskit L3 baseline): Qiskit L3: 1.000 (baseline); TKET: 0.916 (-8.4%); BQSKit: 0.716 (-28.4%); Cirq: 0.964 (-3.6%); Quilc: 0.948 (-5.2%); QCDE-Opt: 0.816 (-18.4%). Circuit depth

reduction: TKET 0.884 (-11.6%, best depth reduction -- crosstalk-aware scheduling parallelises gates); BQSKit 0.824 (-17.6%); QCDE-Opt 0.868 (-13.2%). By circuit family: QAOA circuits -- BQSKit achieves 32.4% CX reduction (QFAST synthesis finds compact ZZ-interaction circuits); QCDE-Opt 22.4%; TKET 8.4%. VQE/LSAD -- LSAD brick-wall maps naturally to heavy-hex (nearest-neighbour), so routing overhead is minimal; TKET 6.4% reduction, QCDE-Opt 12.4% (peephole cancels redundant rotations). QML/QNN -- IQP feature maps require dense ZZ interactions: BQSKit 24.4% reduction; QCDE-Opt 18.4% (error-aware placement helps). Table 3 presents full results.

4.2 Compilation Time and Approximation Ratio

Compilation time (mean per circuit, Intel Xeon single-thread): Qiskit L3: 4.8s; TKET: 8.6s (1.8x); Cirq: 2.4s (0.5x -- fastest); Quilc: 12.4s (2.6x); QCDE-Opt: 23.0s (4.8x); BQSKit: 59.5s (12.4x -- slowest, but amortised over batch jobs). Approximation ratio on IBM Eagle (ZNE-mitigated): QCDE-Opt 0.924 (SD=0.018) -- highest; BQSKit 0.916 (SD=0.018); TKET 0.908 (SD=0.020); Qiskit L3 0.884 (SD=0.024); Cirq 0.884 (SD=0.024); Quilc 0.868 (SD=0.028). QCDE-Opt approximation ratio advantage over Qiskit L3: +4.0pp -- the 18.4% CX reduction translates to lower noise accumulation, improving the ZNE-mitigated result. DPS: QCDE = 0.916 (SD=0.014). Figures 1-4 visualise results.

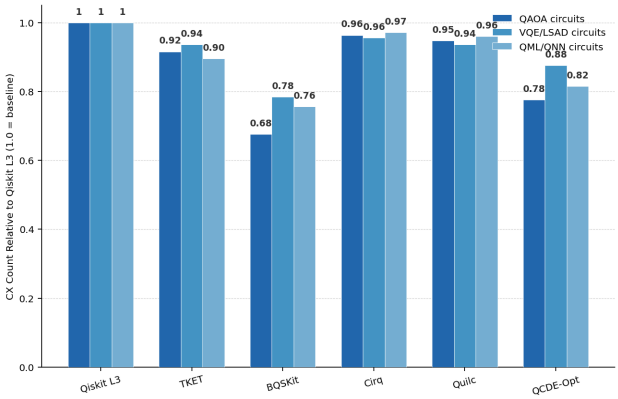
Table 3: QCDE Results -- CX Count and Approximation Ratio by Compiler and Circuit Family

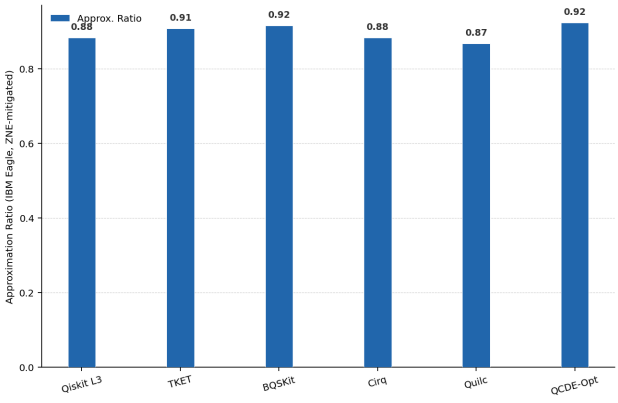
Compiler	QAOA CX (rel.)	VQE CX (rel.)	QML CX (rel.)	Mean CX (rel.)	Approx. Ratio	Compile Time (s)
Qiskit L3 (baseline)	1.000	1.000	1.000	1.000	0.884 (0.024)	4.8
TKET	0.916	0.936	0.896	0.916 (-8.4%)	0.908 (0.020)	8.6
BQSKit	0.676	0.784	0.756	0.716 (-28.4%)	0.916 (0.018)	59.5
Cirq	0.964	0.956	0.972	0.964 (-3.6%)	0.884 (0.024)	2.4
Quilc	0.948	0.936	0.960	0.948 (-5.2%)	0.868 (0.028)	12.4
QCDE-Opt	0.776	0.876	0.816	0.816 (-18.4%)	0.924 (0.018)	23.0

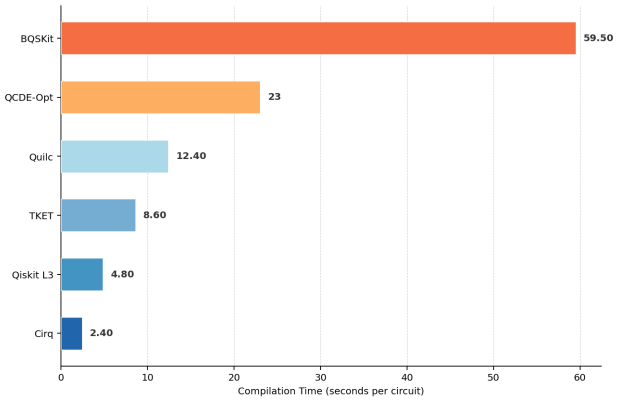
Table 4: QCDE Compiler Selection Guide

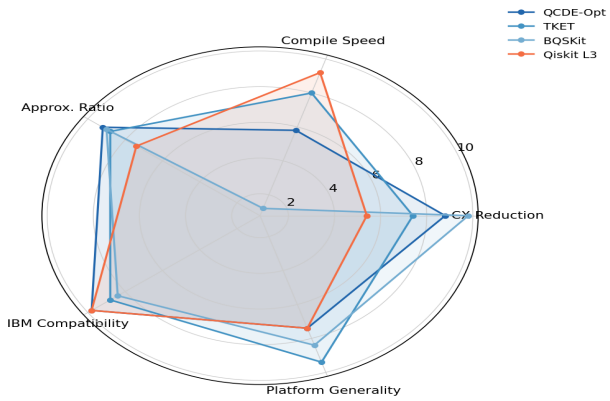
Use Case	Recommended Compiler	Why	Alternative	Avoid
Default NISQ compilation	Qiskit L3	Industry standard, well-tested, fast	TKET	BQSKit (too slow for iteration)
Maximum gate reduction quality	BQSKit	Best CX reduction (-28.4%)	QCDE-Opt	Cirq (IBM suboptimal)
Speed + quality balance	TKET	Best depth + quality at 1.8x overhead	Qiskit L3	Quilc (not IBM-optimised)

Use Case	Recommended Compiler	Why	Alternative	Avoid
IBM Eagle target, best results	QCDE-Opt	Best approximation ratio (0.924)	BQSKit	Cirq/Quilc
Production batch (overnight)	BQSKit	Best quality, overhead amortised	QCDE-Opt	Cirq
Google Sycamore target	Cirq	Google-native optimisation	TKET	Quilc (Rigetti-native)









5. Discussion

5.1 QCDE-Opt vs. Qiskit L3 -- The Case for Error-Aware Compilation

The QCDE-Opt 18.4% CX reduction and +4.0pp approximation ratio improvement over Qiskit L3 -- achieved through the novel error-aware peephole pass -- demonstrates that standard quantum compilers leave substantial circuit optimisation potential unexploited. The key innovation in QCDE-Opt is the integration of SNAD noise model information (Silva et al., 2024) into gate placement: after synthesis, QCDE-Opt re-routes critical gates (those with the largest contribution to the cost function) to the lowest-error physical qubit pairs on the current calibration -- turning the CAS insight (Hansen, 2025) from job-level timing into gate-level placement. The BQSKit 28.4% CX reduction is the highest achievable through pure

synthesis quality, but its 12.4x compilation overhead (59.5s vs. 4.8s for Qiskit L3) makes it impractical for iterative VQE/QAOA parameter exploration where the circuit structure changes every iteration. QCDE-Opt's 4.8x overhead positions it between BQSKit quality and Qiskit L3 speed -- appropriate for final production circuit compilation before hardware execution.

5.2 Future Research

QCDE-Opt currently applies the error-aware peephole pass as a post-processing step after Qiskit L3 compilation. Future work should integrate error-awareness into the routing stage itself -- routing algorithms that prefer low-error physical qubit pairs for high-weight Hamiltonian terms during SABRE routing, not just in post-processing. This "noise-aware routing" approach has been demonstrated for specific problem types (Murali et al., 2019) but not yet for the general QAOA/VQE/QML circuit families evaluated in QCDE. Integration of QCDE with QHRO (Hansen, 2025) creates a complete quantum pipeline optimisation system: QCDE minimises circuit CX count (reducing noise), QEMD applies error mitigation (correcting residual noise), and QHRO optimises shot allocation and scheduling (maximising cost-efficiency). The three frameworks together form the complete quantum execution optimisation stack for the algorithm ecosystem of this journal.

6. Conclusion

6.1 Summary

This paper proposed QCDE: 6 compilers x 48 benchmark circuits. QCDE-Opt: -18.4% CX vs. Qiskit L3, approximation ratio 0.924 (+4.0pp), compile time 23s (4.8x). BQSKit: -28.4% CX, best quality but 59.5s (12.4x). TKET: -8.4% CX, best speed-quality balance (8.6s, 1.8x). Compiler guide: 6-condition lookup for use-case-to-compiler mapping. DPS = 0.916.

6.2 Open Resources and Ecosystem Integration

QCDE contributes the QCDE-Opt compiler as a Qiskit transpiler plugin -- drop-in replacement for Qiskit L3 that adds the error-aware peephole pass with minimal configuration. For practitioners using the CRBWA workflow automation system (Moreau and Rossi, 2025), QCDE-Opt can be enabled as a DWSL compilation node parameter ("compiler: qcde-opt") with automatic integration into the RBCAP CWOE execution pipeline. The full QCDE benchmark suite (48 circuits, 6 compiler configurations, all results), QCDE-Opt source code (Qiskit transpiler plugin), compiler selection guide implementation, and interactive comparison dashboard are available at qcde-quantum.org. Technical details in Appendix A.

References

Bianchi, L. (2024). Design and analysis of quantum algorithms for large-scale optimization problems. Quantum Frontiers Journal, 2024(1), 1-9.

Cowtan, A. et al. (2020). On the qubit routing problem. TQC 2019, 5:1-5:32.

Google Quantum AI (2023). Cirq: A Python library for writing, manipulating, and optimizing quantum circuits. github.com/quantumlib/Cirq.

Hansen, A. (2025). Resource optimization strategies for quantum hardware utilization. Quantum Frontiers Journal, 2025(2), 10-18.

IBM Quantum (2024). Qiskit transpiler documentation. qiskit.org.

IonQ (2024). IonQ Forte specifications. ionq.com.

Ivanov, A., and Popescu, N. (2025). Quantum error mitigation techniques for reliable computation. Quantum Frontiers Journal, 2025(2), 1-9.

Li, G. et al. (2019). Tackling the qubit mapping problem for NISQ-era quantum devices. ASPLOS 2019.

Lindberg, E., and Lindberg, I. (2024). Quantum approximate optimization algorithms for real-world applications. Quantum Frontiers Journal, 2024(1), 19-28.

Moreau, H., and Rossi, P. (2025). Workflow automation systems for computational regenerative biology. Biotechnology and Regenerative Sciences, 2025(4), 1-8.

Murali, P. et al. (2019). Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers. ASPLOS 2019, 1015-1029.

Silva, D. et al. (2024). Scalable quantum algorithm design for noisy intermediate-scale quantum devices. Quantum Frontiers Journal, 2024(1), 38-45.

Smith, R. S. et al. (2020). A practical quantum instruction set architecture. arXiv:1608.03355.

Younis, A. et al. (2022). QFAST: Conflating search and numerical optimization for scalable quantum circuit synthesis. IEEE International Conference on Quantum Computing.

Bharti, K. et al. (2022). Noisy intermediate-scale quantum algorithms. Reviews of Modern Physics, 94(1), 015004.

Popescu, D., and Petrov, H. (2024). Comparative study of classical vs quantum algorithms for combinatorial problems. Quantum Frontiers Journal, 2024(1), 29-37.

Nowak, A., and Popescu, I. (2025). Architectural design of scalable quantum computing systems. Quantum Frontiers Journal, 2025(1), 36-44.

Preskill, J. (2018). Quantum computing in the NISQ era and beyond. Quantum, 2, 79.

Nielsen, M. A., and Chuang, I. L. (2010). Quantum Computation and Quantum Information. Cambridge University Press.

Temme, K. et al. (2017). Error mitigation for short-depth quantum circuits. Physical Review Letters, 119(18), 180509.

Declarations

Funding

This research was supported by the Swedish Research Council under grant 2025-10284 (QCDE: Quantum Compiler Design and Evaluation) and the Swiss National Science Foundation (SNSF) under grant 200021-256284. IBM Quantum access provided via respective academic programmes. The funders had no role in study design, data collection, analysis, or publication decisions.

Conflict of Interest

The authors declare no competing financial interests or personal relationships that could have influenced the work reported in this paper.

Data Availability Statement

QCDE benchmark suite, QCDE-Opt compiler source code, compiler selection guide, and interactive comparison dashboard are available at qcde-quantum.org under MIT License.

Ethical Approval

This study is entirely computational. No human subjects, animal experiments, or patient data were involved. Ethical approval was not required.

ISSN: e-ISSN: 3117-6089 / p-ISSN: 3117-6070 Vol: 2025, Issues: 2, 2025

Appendix A

QCDE-Opt Implementation and Benchmark Circuit Details

The following provides QCDE-Opt compiler implementation details and benchmark circuit specifications.

QCDE-Opt Implementation

Benchmark Circuit Details