Application Driven Evaluation of Fault-Tolerant Quantum Computing Architectures

Kevin Obenland

IPAM Workshop
2/19/2026

This material is based upon work supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Under Secretary of Defense for Research and Engineering.

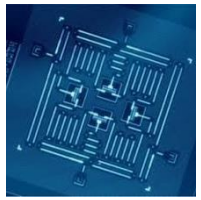
LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Analyzing NISQ and Fault-Tolerant Architectures

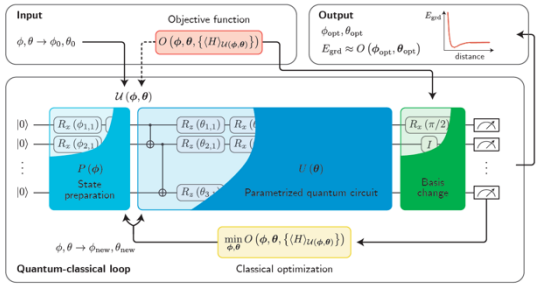
NISQ Technologies

Google Sycamore

Transmon qubit



VQE Algorithm(1)



Circuit Compilation

Algorithm Design

Resource Estimation

Architecture Design

HW/SW Codesign

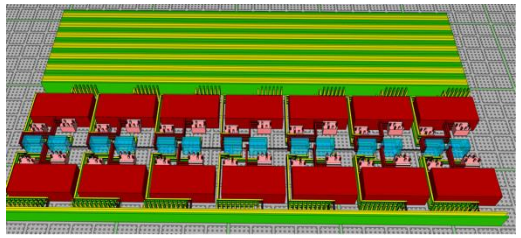
Hierarchical System Design

Classical HPC Processing

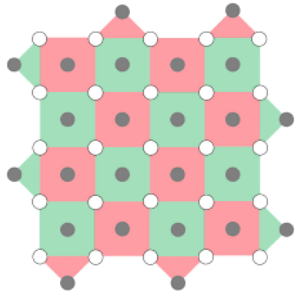
Classical Control Systems

Fault-tolerant QC Technologies

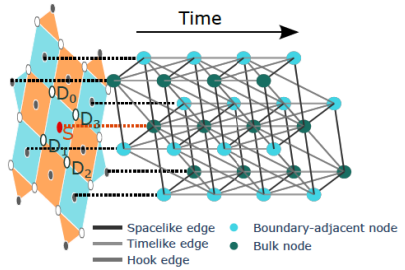
Factoring Machine Architecture (2)



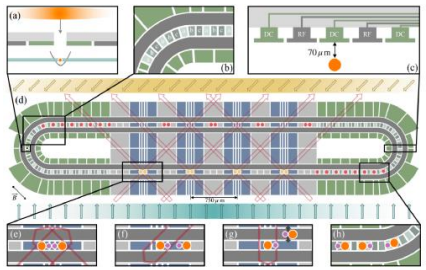
Surface Codes



Syndrome Decoders (3)



Quantinuum Race-Track Architecture



IBM Bluejay Concept

Shared need for Design, Analysis, and Classical tools across NISQ and FT platforms (even though the details may be different)

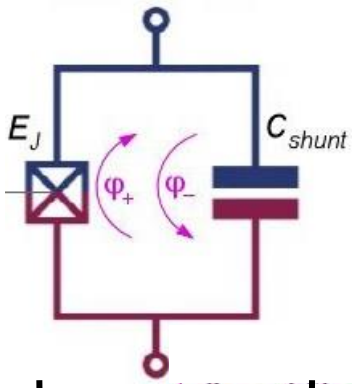
1) Noisy intermediate-scale quantum (NISQ) algorithms. arXiv: 2101.08448
2) G.Gidney, A.Fowler. “Flexible layout of surface code computations using AutoCCZ states.” arXiv:1905.08916
3) A real-time, scalable, fast and highly resource efficient decoder for a quantum computer. arXiv: 2309.05558

Outline

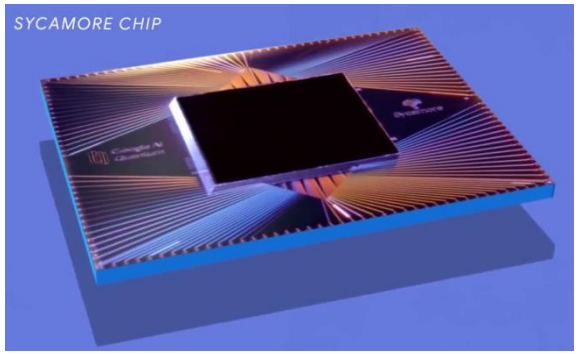
- Architecture Analysis of near-term architectures
- Generating logical circuits for utility-scaled applications
  - Mapping to quantum processors
  - Generating circuits with pyLIQTR
- Analyzing Fault-tolerant architectures
  - Gate counting to determine algorithmic scaling
  - Circuit scheduling to emulate the impact of hardware constraints
- Open questions in quantum architectures

Near-term Quantum Computing Architectures

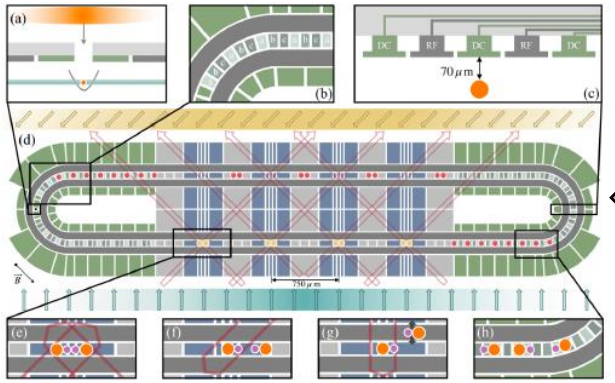
Transmon Qubit



Sycamore Processor



Quantinuum H2 Architecture



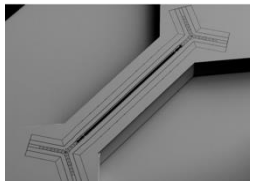
Device Optimization → **Early Architectures** → **Optimized Architectures**

Focus of our work:
1. Comparing performance of current architectures
2. Optimizing future architectures

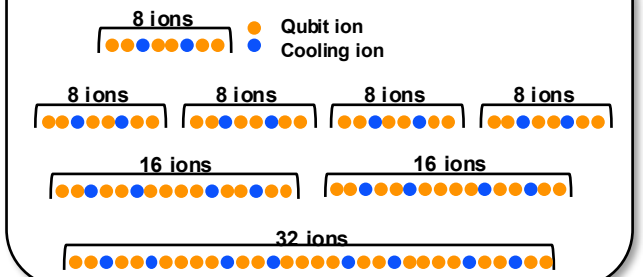
Benchmarking current and emerging architectures will enable optimization of future designs

Analyzing Trapped-Ion Architectures (1)

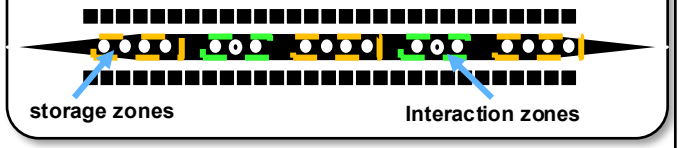
Arch. #1: Fully Connected Trap (Single)



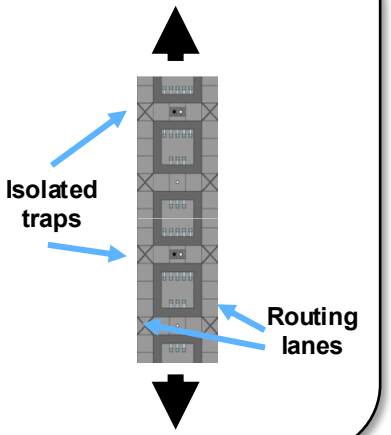
Arch. #2: Reconfigurable Multicore QA (RMQA)



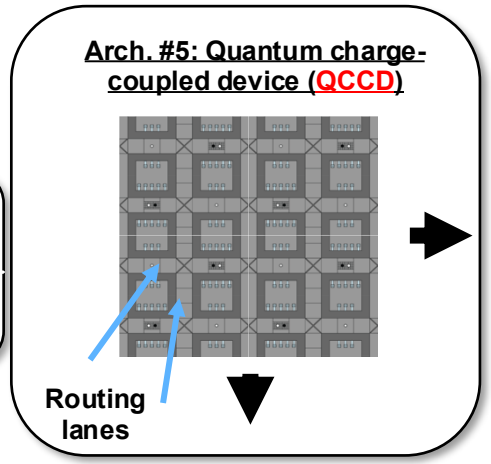
Arch. #3: Single Trap, Multi-Zoned



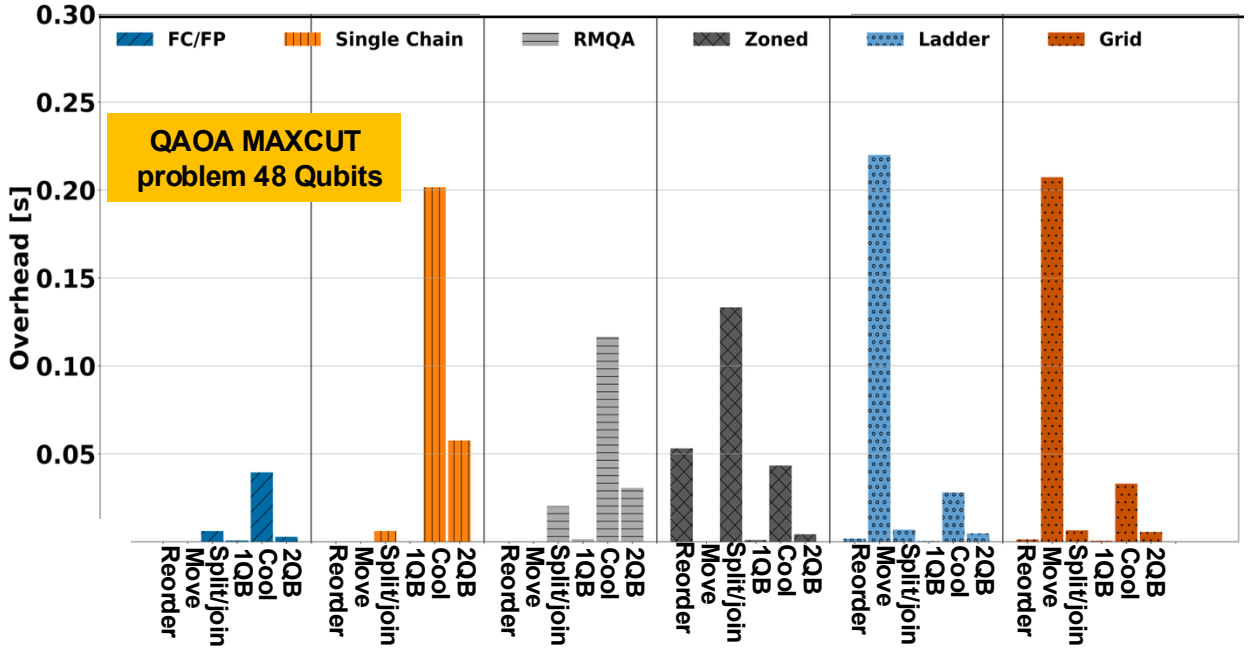
Arch. #4: Multi-Trap Ladder



Arch. #5: Quantum charge-coupled device (QCCD)



Critical Operations in Compiled Circuit



The operations that are critical depend on the details of the architecture

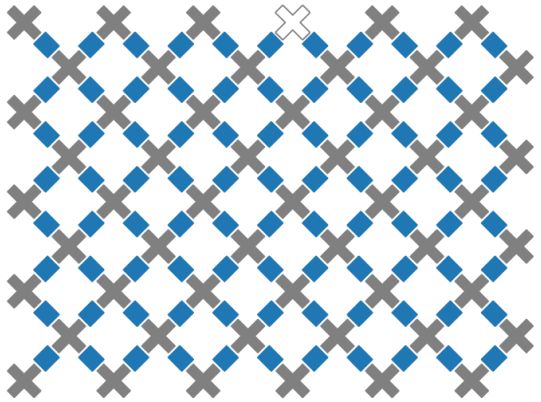
1) A.Kurlej, et.al. "Benchmarking and Analysis of Noisy Intermediate-Scale Trapped Ion QC IEEE QCE 2022

Arch#1: https://www.nature.com/articles/nature18648. https://www.osti.gov/biblio/1237003
Arch #2: https://ionq.com/posts/august-25-2021-deep-dive-reconfigurable-multicore-quantum-architecture

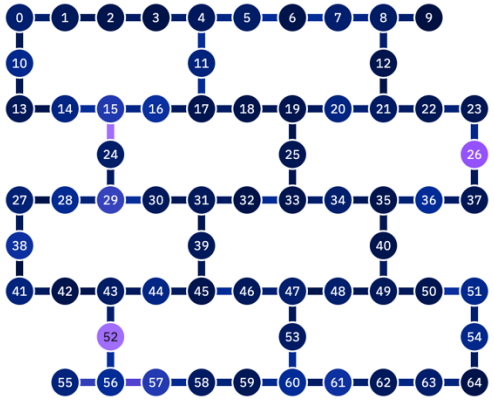
Arch#3: https://journals.aps.org/prx/pdf/10.1103/ PhysRevX.7.041061
Arch #4,5: https://www.nature.com/articles/ nature00784

Analysis of QAOA on Superconducting Qubit Architectures (1)

Google Sycamore[1]

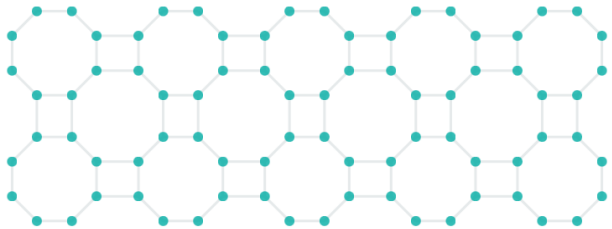


IBM Ithaca[2]

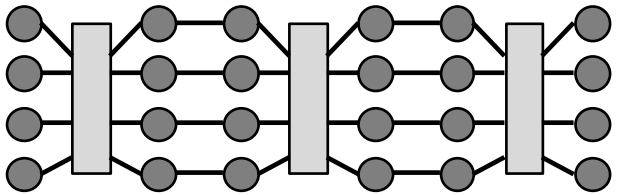


Commercially available architectures and envisioned future ones

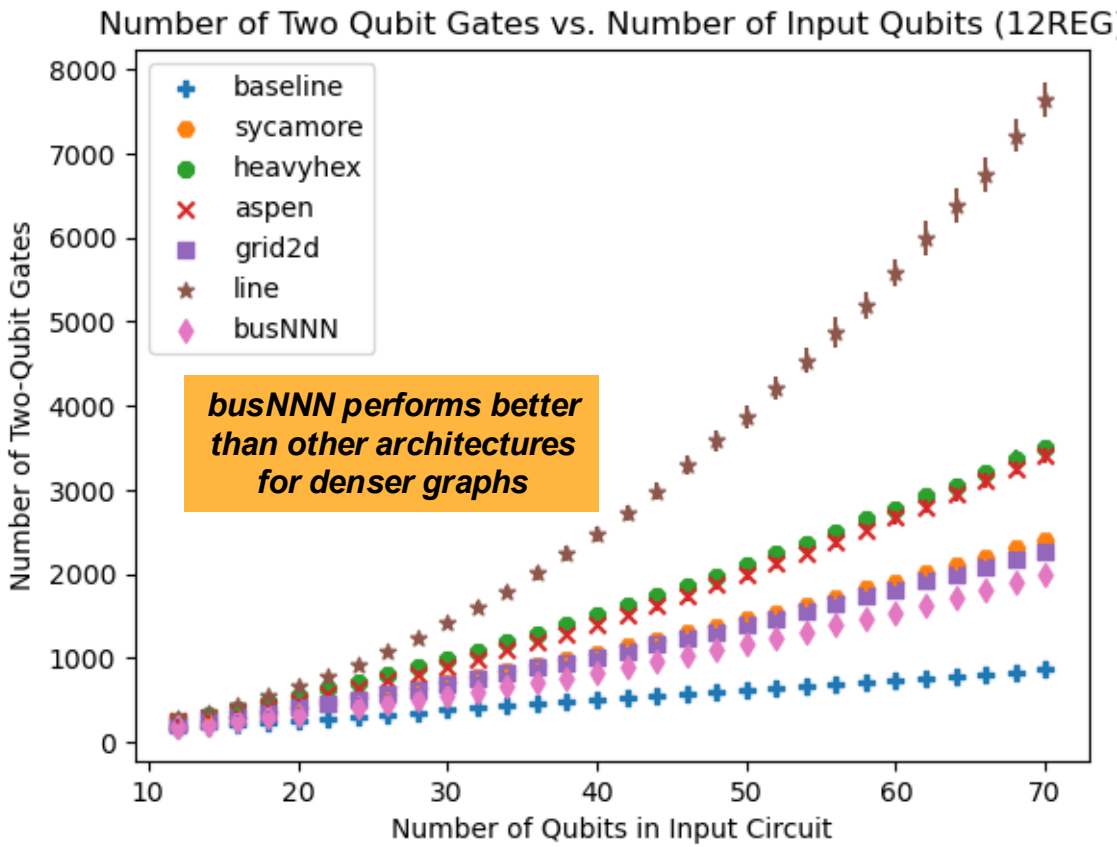
Rigetti Aspen M3[3]



busNNN Architecture



Bus size = 8, Total busses = 3



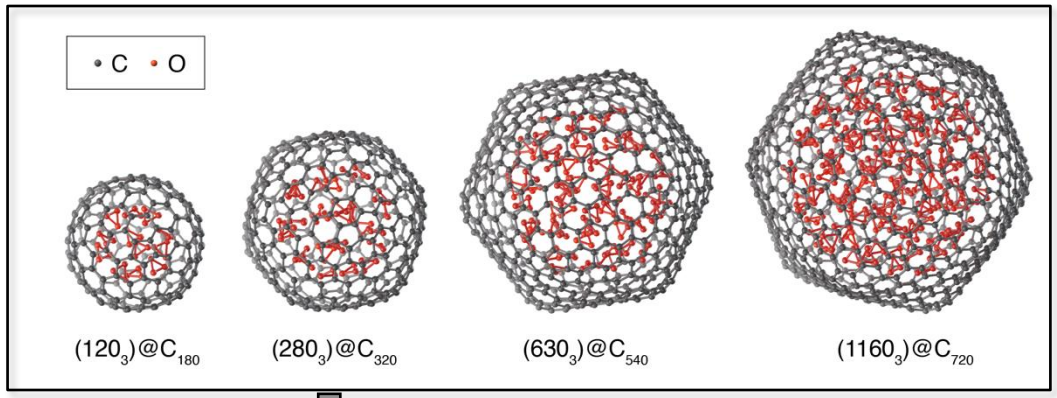
1) B. Rempfer, et.al. "Comparison of superconducting NISQ architectures." IEEE QCE 2023

1. https://quantumai.google/cirq/google/devices
2. https://quantum-computing.ibm.com/services/resources?services=systems&limit=50
3. https://qcs.rigetti.com/qpus

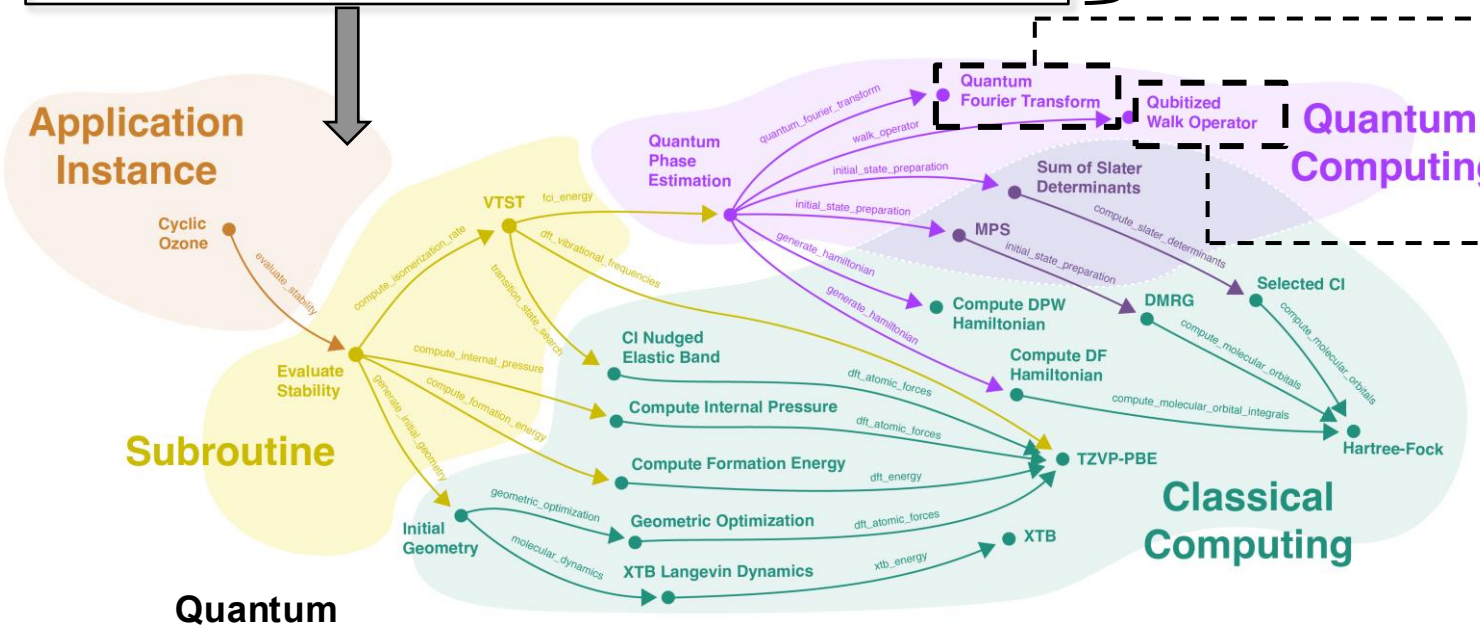
Outline

- Architecture Analysis of near-term architectures
- **Generating logical circuits for utility-scaled applications**
  - **Mapping to quantum processors**
  - **Generating circuits with pyLIQTR**
- Analyzing Fault-tolerant architectures
  - Gate counting to determine algorithmic scaling
  - Circuit scheduling to emulate the impact of hardware constraints
- Open questions in quantum architectures

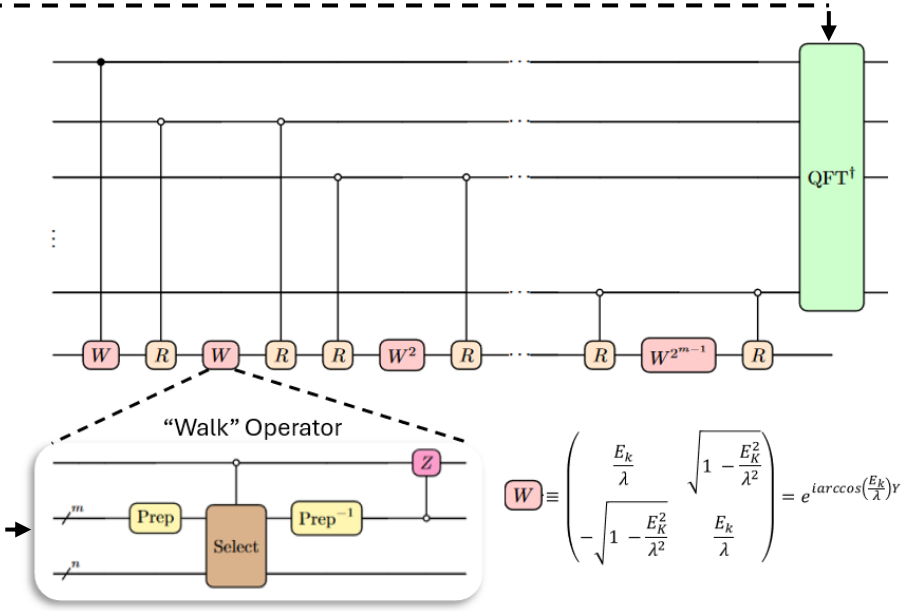
Partitioning Application Instances (Cyclic Ozone[1])



Geometries for ozone-containing endofullerenes. Geometries were obtained, and optimized, using Density Functional Theory (DFT)

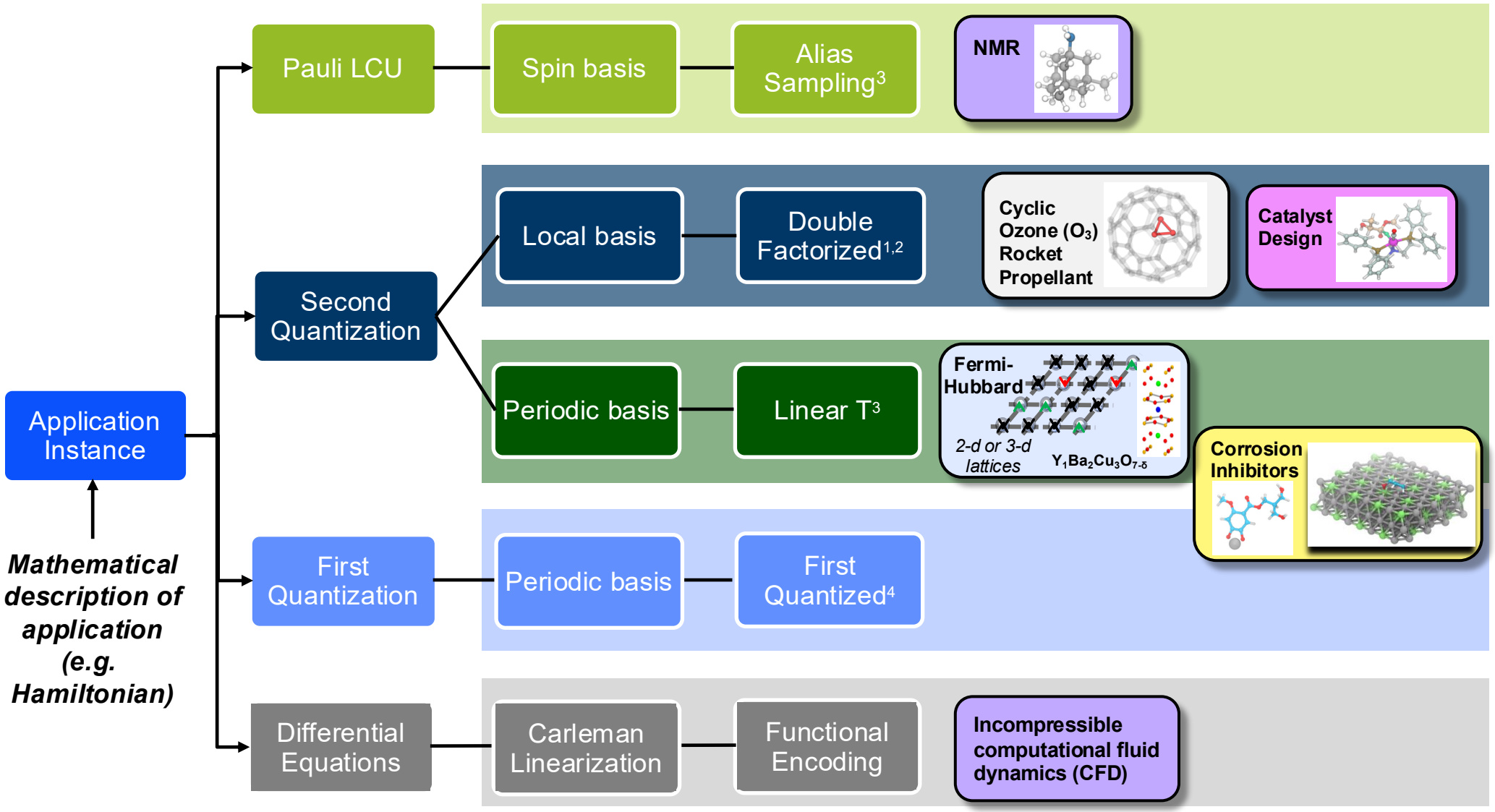


Quantum Benchmarking Graph



[1] *T.W. Watts, et. al., "Fullerene-encapsulated Cyclic Ozone for the Next Generation of Nano-sized Propellants via Quantum Computation". arXiv:2408.13244*

Encoding and Implementing Applications



Choice of encoding (i.e. implementation method) depends on details of problem

Framework provides a number of different methods to accommodate a wide variety of problems

pyLIQTR (Opensource SW for Generating Circuit Implementations)

Library Structure

- pyLIQTR
  - ProblemInstance
    - Chemical Ham.
    - Electronic Struct.
    - Lattice
  - Block Encodings
    - DoubleFactorized
    - LinearT
    - Carleman
  - Algorithms
    - QSP/QSVT
    - GSEE, Dynamics
  - Classical prep
    - Angle Gen.
    - PEST
    - Clifford+T syn.

Hamiltonian creation

Block Encoding and Circuit creation

Algorithm Circuit Implementation

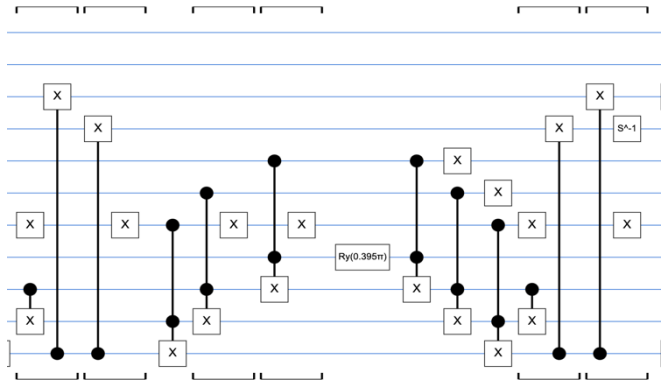
Phase angle generation, rotation gate synthesis, etc.

pyLIQTR Version 1.4.2

pip install pyliqtr

pyLIQTR (LIncoln laboratory Quantum algorithm Test and Research) is a python-based tool-kit that contains implementation of quantum algorithms, classical pre-processing, circuit generation, and resource analysis.

```
from pyLIQTR.ProblemInstances.getInstance  import *
from pyLIQTR.BlockEncodings.getEncoding    import *
from pyLIQTR.utils.resource_analysis       import estimate_resources
from pyLIQTR.utils.circuit_decomposition   import circuit_decompose_multi
from pyLIQTR.qubitization.phase_estimation import QubitizedPhaseEstimation
from pyLIQTR.qubitization.qubitized_gates  import QubitizedWalkOperator
from pyLIQTR.utils.printing                import openqasm
```

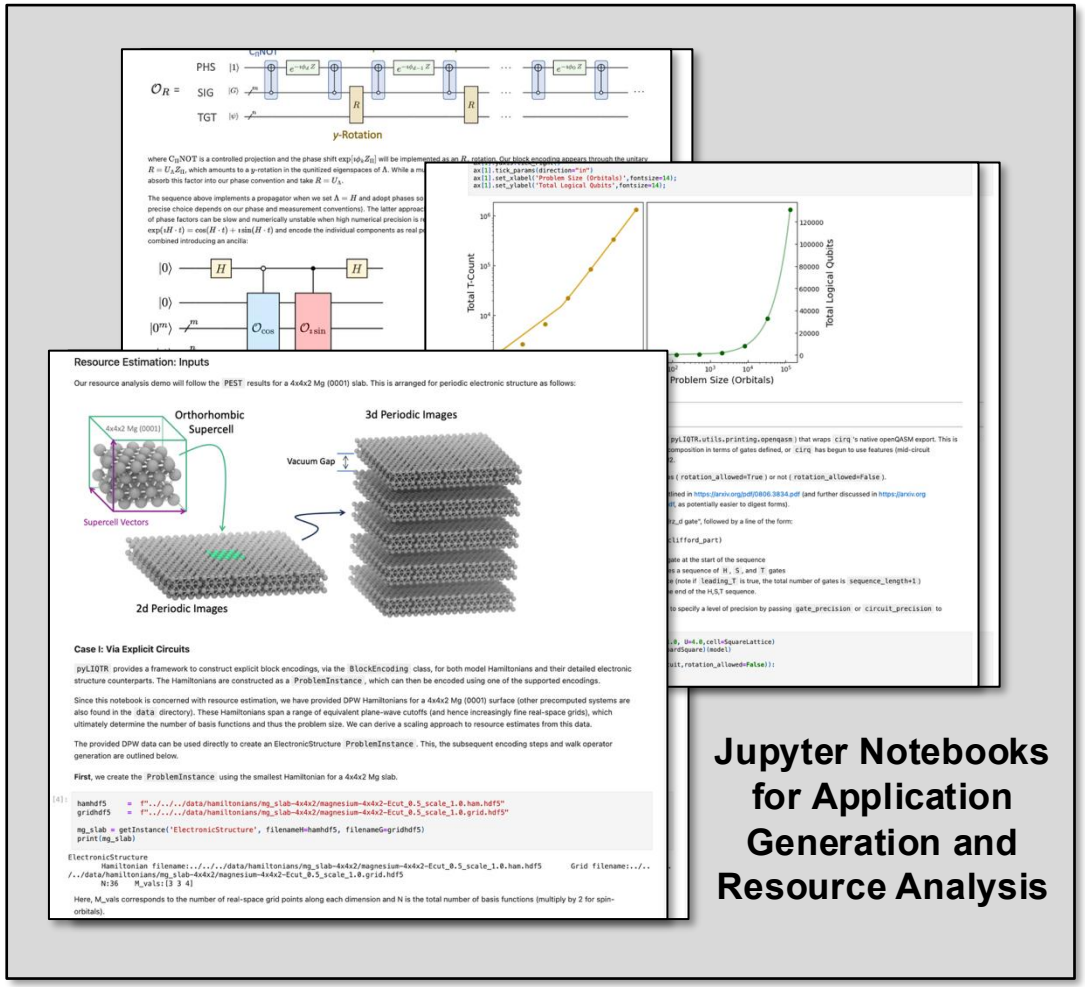


Logical Circuit Implementation in Google Cirq and Qualtran

quantumlib/
Qualtran

Application Problems (pyLIQTR Notebooks)

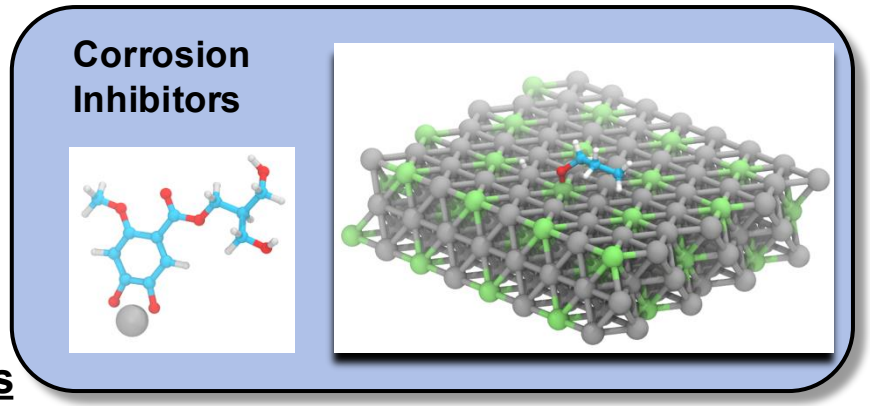
- Homogeneous catalysis
  - Double-factorized Block encoding
- Fermi Hubbard
  - Linear-T block encoding
- Local Chemistry
  - Pauli-LCU block encoding
- Periodic Chemistry
  - Linear-T electronic structure encoding
- Non-Linear ODE
  - Carleman linearization
- Heisenburg
  - Pauli-LCU block encoding



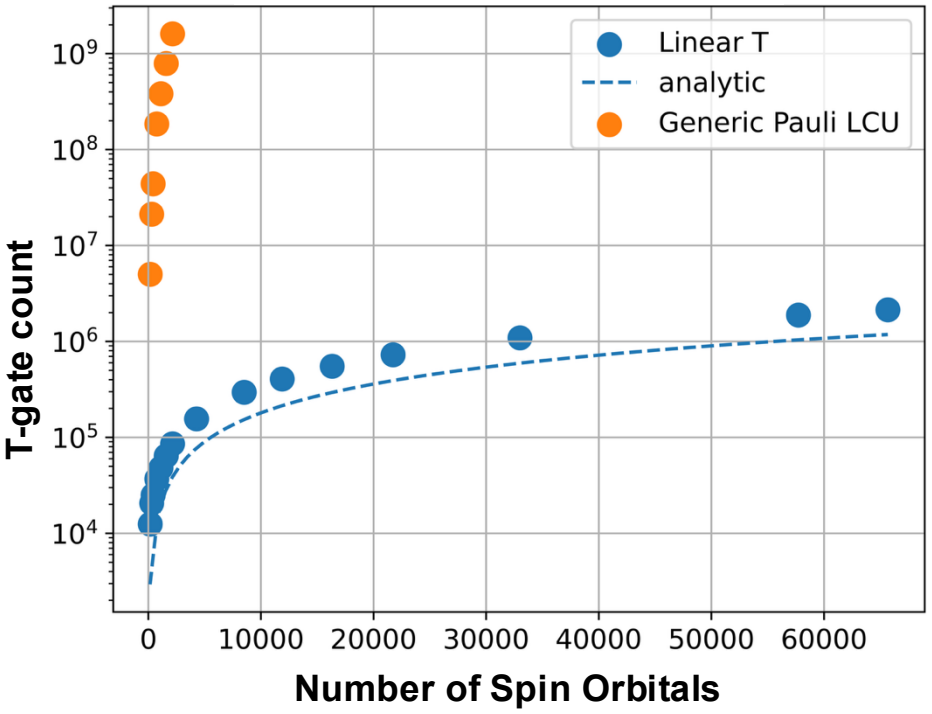
Jupyter Notebooks for Application Generation and Resource Analysis

Block Encoding Comparison: Magnesium Slab (Resources)

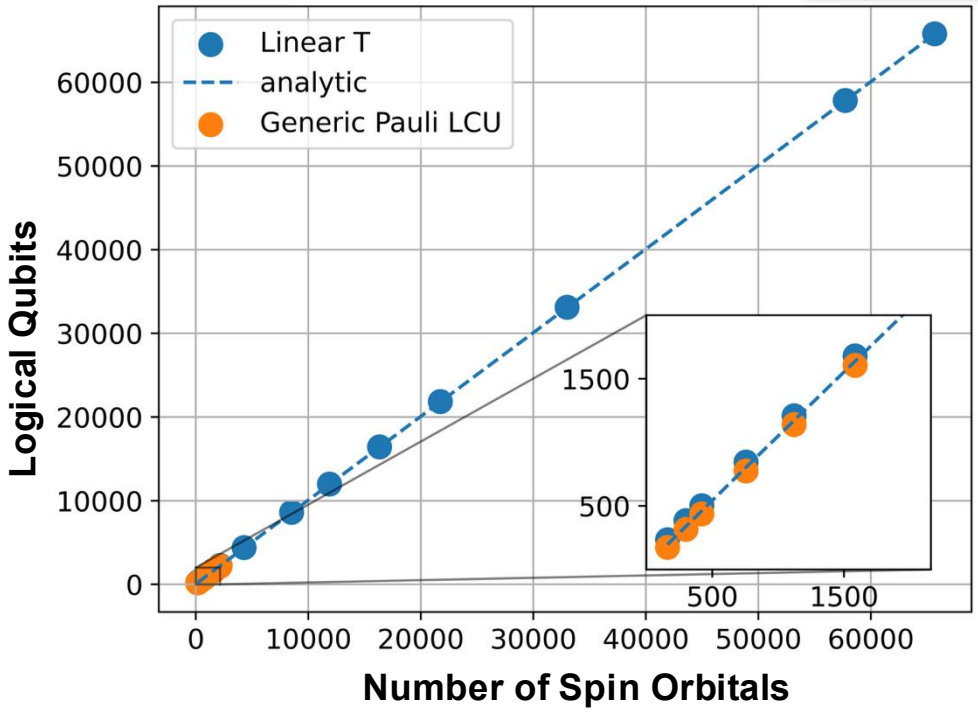
- Linear T encoding reduces T gate count by many orders of magnitude
- Increase in qubit count is modest



T-count



Number of Qubits

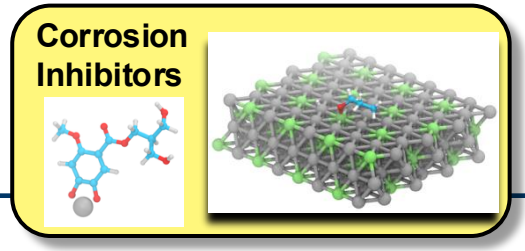


Linear T encoding takes two orders of magnitude less time to execute than Pauli LCU

Linear T encoding assumes a phase estimation energy error $\Delta E = 10^{-3}$ and approximation bits $\lceil \log(\frac{2\sqrt{2}\lambda}{\Delta E}) \rceil$

QPE circuit rotation synthesis error budget of 10^{-4}

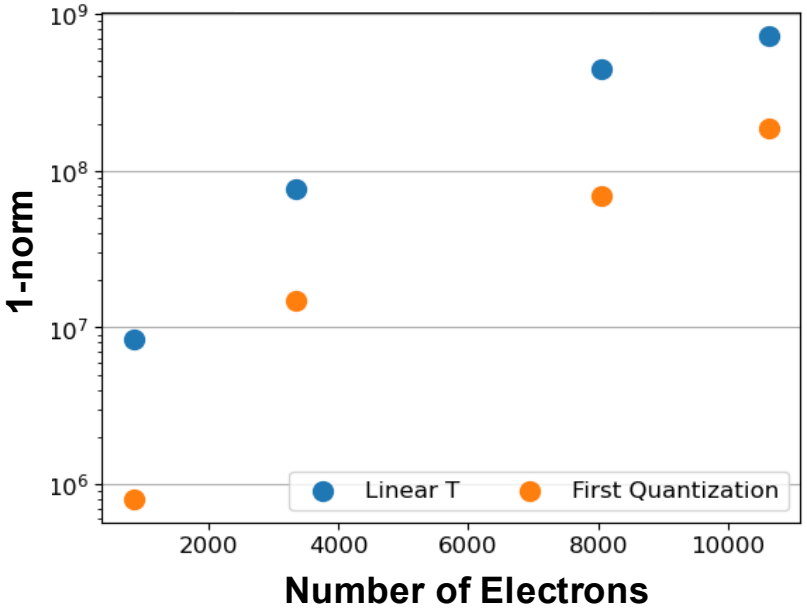
Circuit Results for Qubitized Phase Estimation



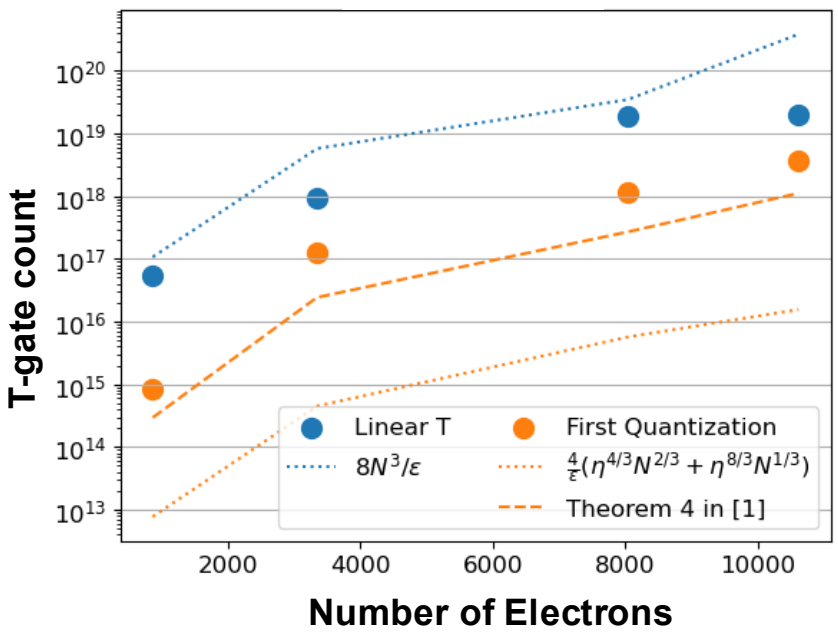
Encodings assume a phase estimation energy error $\Delta E = 10^{-3}$

Results for target energy cutoff of 40 Ry

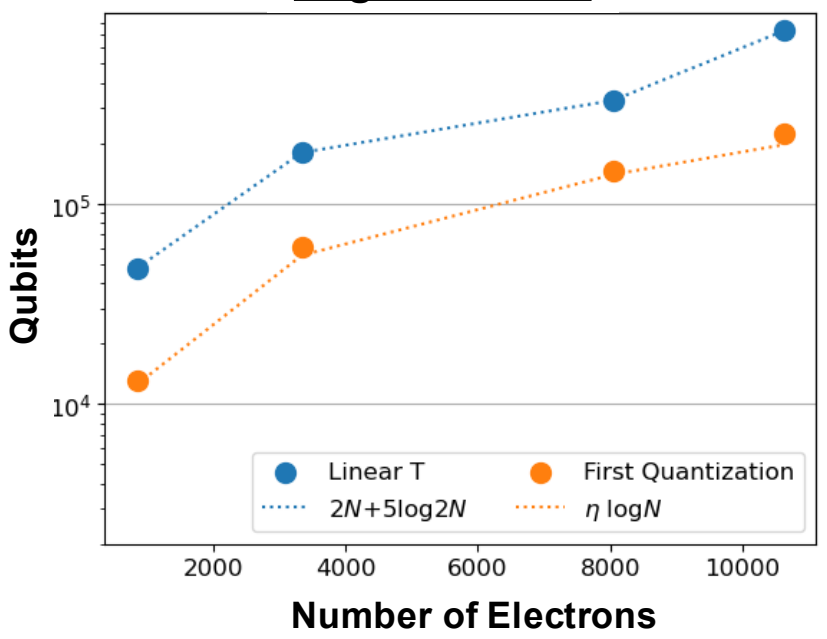
Hamiltonian Norm



T-count



Logical Qubits



- **Asymptotic expressions for qubit count match well with circuit estimates**
- **First quantized qubit count ~3x lower and T-gate count ~10-100x lower than linear T**
- **Details of circuit composition important for gate counts**

First quantized encoding reduces qubit count by 10x and T count by up to 100x

[1]Su, et. al., arxiv:2105.12767

Outline

- Architecture Analysis of near-term architectures
- Generating logical circuits for utility-scaled applications
  - Mapping to quantum processors
  - Generating circuits with pyLIQTR
- **Analyzing Fault-tolerant architectures**
  - **Gate counting to determine algorithmic scaling**
  - **Circuit scheduling to emulate the impact of hardware constraints**
- Open questions in quantum architectures

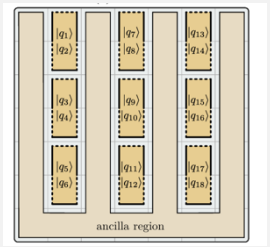
Architecture Models for Resource Estimation

Gate Counting

- Circuit decomposed into Clifford+T gates
  - Typically only T gates are reported
- Counts based on analytical formulas or explicit circuits
- Gives a rough estimate of scaling
- Repeated blocks only need counting once

pyLIQTR estimate_resources

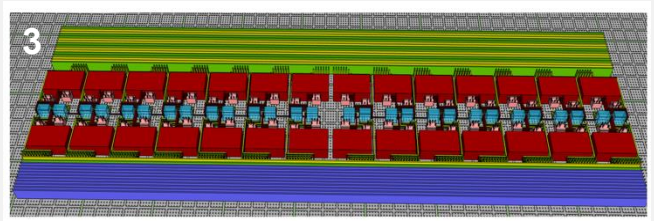
Game of Surface Codes[1]



- Gates executed via multi-qubit measurements
- Clifford gates pushed to end of circuit and folded into measurements
- Serial execution of T gates

MS Azure Resource Estimation[2]

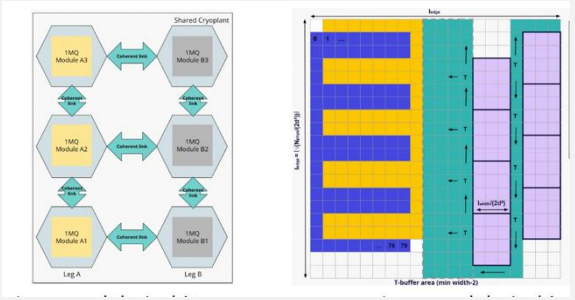
Distributed Lattice Surgery



- Communication via lattice surgery
- Computation is distributed
- Clifford gates are executed inline

pyLIQTR scheduler

Graph-State Compilation

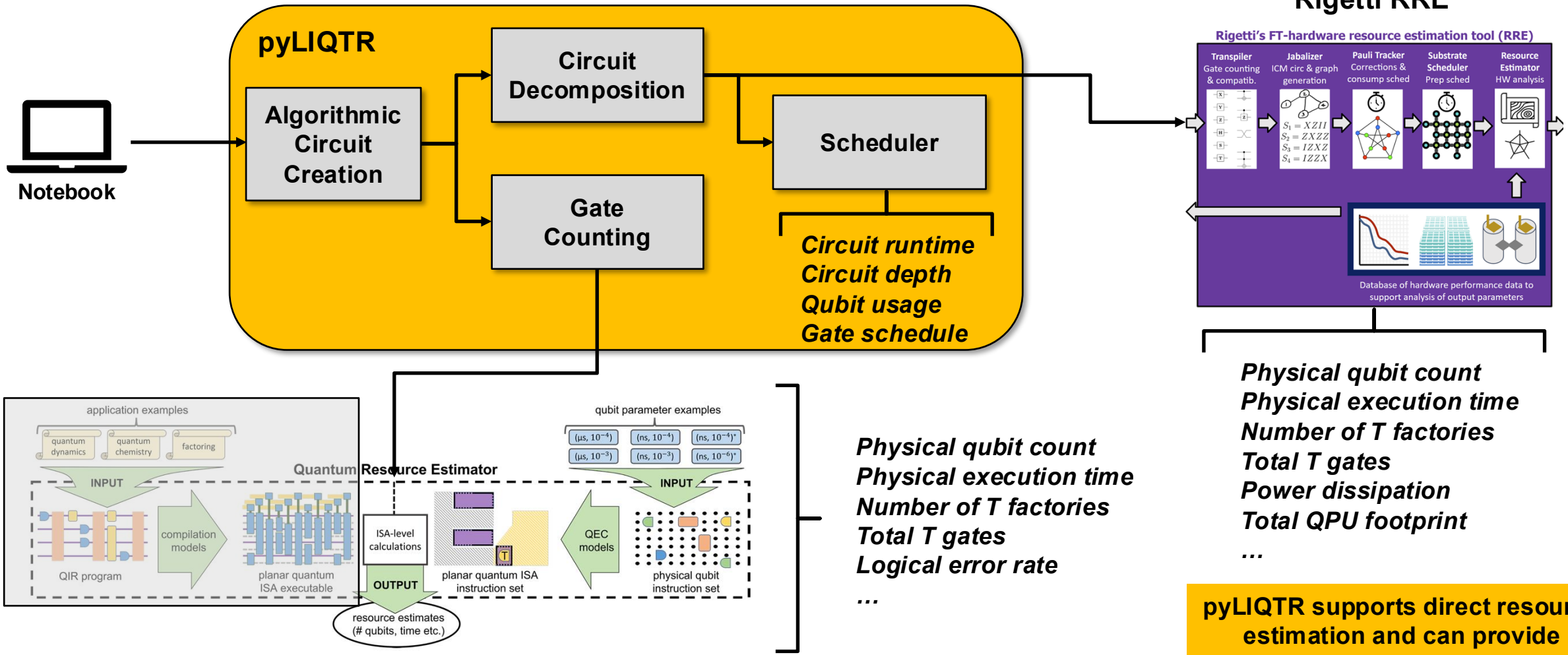


- Circuit compiled into:
  1) Graph-state creation
  2) Graph consumption
- Compiles onto a measurement-based architecture

RRE[4] (Rigetti)

1. D. Litinski, "A Game of Surface Codes". Quantum 3 ,128 (2019)
2. M. E. Beverland,et.al."Assessing requirements to scale to practical quantum advantage," 2022, arXiv:2211.07629.
3. C.Gidney, M.Ekera. "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits". Quantum 5, 433 (2021)
4. S,N.Saadatmand, et.al. Fault-tolerant resource estimation using graph-state compilation on a modular superconducting qubit architecture. arXiv:2406.06015

Using pyLIQTR for Logical & Physical Resource Estimation

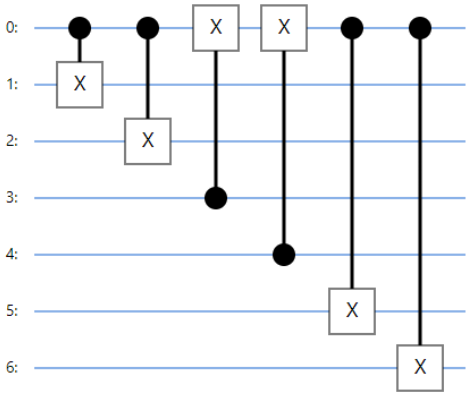


pyLIQTR supports direct resource estimation and can provide circuits for external tools

1. M. E. Beverland,et.al."Assessing requirements to scale to practical quantum advantage," 2022, arXiv:2211.07629.
2. S,N.Saadatmand, et.al. Fault-tolerant resource estimation using graph-state compilation on a modular superconducting qubit architecture. arXiv:2406.06015

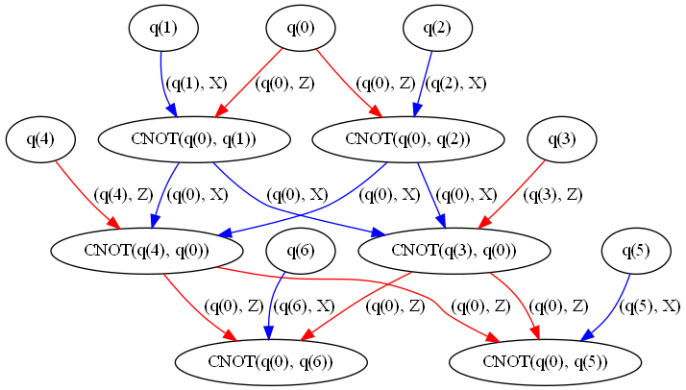
Time-based Scheduling in pyLIQTR

Input Logical Circuit



Circuit gets compiled to Directed Acyclic Graph (DAG)

Dependency DAG



Traverse DAG, scheduling ready instructions in time order

Output Analysis

Execution time: 12
Circuit T-depth: 0
Qubits: 7
Gate profile: {"T": 0, "CX": 6, "Pauli":0}

Customizable parameters:

Resource and architectural specifications: specifies timing for gate types and structural restrictions on execution of T-gates
Decomposition level: Controls the degree of decomposition of circuit
Decomposition protocol: Controls how repeated components are accumulated
Gateset: specifies which gates to include in analysis

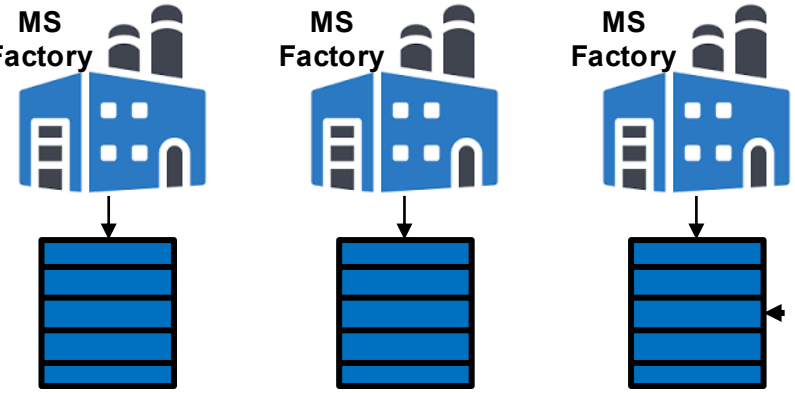
A time-based scheduler provides a cycle-by-cycle schedule of gate execution

This can be used for estimation or to drive a cycle-accurate logical simulation

Scheduler results:

Circuit execution time: based on operation timings and ordering, total circuit time
Circuit T-depth: total number of circuit 'moments' which contain at least one T-gate
Number of qubits used: total number of logical qubits
Gate profile: a full tally of basic gates in the circuit.
Active qubit distribution: Distribution of active qubits
T-widths: a distribution of the number of T-gates contained within each time step

Resource Estimation with the pyLIQTR Scheduler



Cycles to create one state

Cycles required to apply state

Acceptance probability

```
#Build T-factories
t_factory_1 = state_factory(production_time=4, consumption_time=1, production_limit=2, success_prob=0.8)
t_factory_2 = state_factory(production_time=4, consumption_time=1, production_limit=2, success_prob=0.8)
```

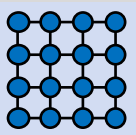
Number that can be buffered

Magic State Factory Parameters(1)

Protocol	p_{phys}	p_{out}	Qubits	Cycles	Space-time cost per output state	
					Qubitcycles	Full distance
$(15\text{-to-}1)_{7,3,3}$	10^{-4}	4.4×10^{-8}	810	18.1	14,600	$5.49d^3/d = 11$ | $3.33d^3/d = 13$
$(15\text{-to-}1)_{9,3,3}$	10^{-4}	9.3×10^{-10}	1,150	18.1	20,700	$4.71d^3/d = 13$ | $3.07d^3/d = 15$
$(15\text{-to-}1)_{11,5,5}$	10^{-4}	1.9×10^{-11}	2,070	30.0	62,000	$9.19d^3/d = 15$ | $6.31d^3/d = 17$
$(15\text{-to-}1)^4_{9,3,3} \times (20\text{-to-}4)_{15,7,9}$	10^{-4}	2.4×10^{-15}	16,400	90.3	371,000	$27.0d^3/d = 19$ | $20.0d^3/d = 21$
$(15\text{-to-}1)^4_{9,3,3} \times (15\text{-to-}1)_{25,9,9}$	10^{-4}	6.3×10^{-25}	18,600	67.8	1,260,000	$25.9d^3/d = 29$ | $21.2d^3/d = 31$
$(15\text{-to-}1)_{17,7,7}$	10^{-3}	4.5×10^{-8}	4,620	42.6	197,000	$6.30d^3/d = 25$ | $4.04d^3/d = 29$
$(15\text{-to-}1)^6_{13,5,5} \times (20\text{-to-}4)_{23,11,13}$	10^{-3}	1.4×10^{-10}	43,300	130	1,410,000	$28.9d^3/d = 29$ | $19.6d^3/d = 33$
$(15\text{-to-}1)^4_{13,5,5} \times (20\text{-to-}4)_{27,13,15}$	10^{-3}	2.6×10^{-11}	46,800	157	1,840,000	$30.9d^3/d = 31$ | $21.5d^3/d = 35$
$(15\text{-to-}1)^6_{11,5,5} \times (15\text{-to-}1)_{25,11,11}$	10^{-3}	2.7×10^{-12}	30,700	82.5	2,540,000	$35.3d^3/d = 33$ | $25.0d^3/d = 37$
$(15\text{-to-}1)^6_{13,5,5} \times (15\text{-to-}1)_{29,11,13}$	10^{-3}	3.3×10^{-14}	39,100	97.5	3,810,000	$37.6d^3/d = 37$ | $27.7d^3/d = 41$
$(15\text{-to-}1)^6_{17,7,7} \times (15\text{-to-}1)_{41,17,17}$	10^{-3}	4.5×10^{-20}	73,400	128	9,370,000	$39.8d^3/d = 49$ | $31.5d^3/d = 53$

Factory parameters specified as a table. Table from (1) used as example, but specification is arbitrary

qubit_block



Block parameters

```
#to assign specific qubits to blocks:
qb_0 = qubit_block(p=4, m=4, f=4, t=1, qubit_list=['target0', 'target1', 'target2', 'target3'], factories=[t_factory_1])
qb_1 = qubit_block(p=3, m=2, f=8, t=1, qubit_list=['selection0', 'selection1', 'selection2', 'selection3', 'selection4'], factories=[t_facto
qb_2 = qubit_block(p=3, m=5, f=2, t=1, qubit_list=['gancilla_c(0)', 'gancilla_c(1)'], factories=[t_factory_2])
qb_3 = qubit_block(p=4, m=2, f=6, t=1, qubit_list=['gancilla_c(2)', 'gancilla_c(3)'], factories=[t_factory_2])
```

Gate parameters

Qubit allocation

Factory allocation

Defines code distance/timing, and #physical qubits

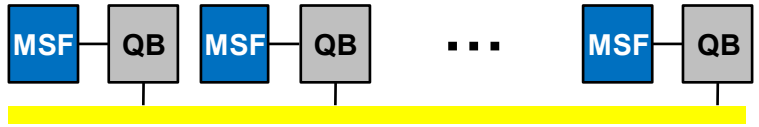
```
def define_surface_code(p_err=1e-3, pth=0.01, alpha= 0.03, logical_cycle_time=1):
    qecCode = PhysicalQECCode(p_physical = p_err)

    for dist in range(3,51+1,2):
        logical_gate_cycles = {'Idle' : dist, 'T' : dist, 'Clifford' : dist}
        code = {'CodeParms': (dist*dist, 1, dist), 'QubitsPerBlock': 2*dist*dist-1}
        qecCode.addCode(p=p_err, code_parameters=code, logical_cycle_time=logical_cycle_time,
                        logical_gate_cycles=logical_gate_cycles, pth=pth, alpha=alpha, code_type='RotatedSurface')
```


1) D.Litinski. "Magic State Distillation: Not as Costly as you Think." Quantum 3, 205 (2019)

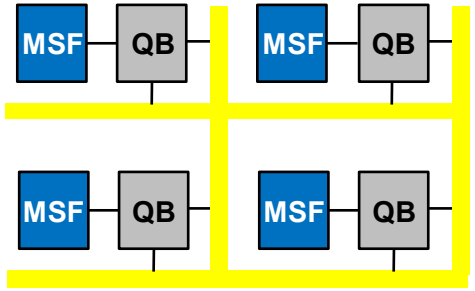
Architecture Models Considered

Linear with Private Factories

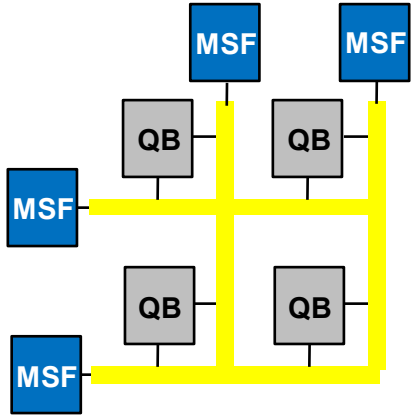


Linear with Shared Factories

Grid with Private Factories



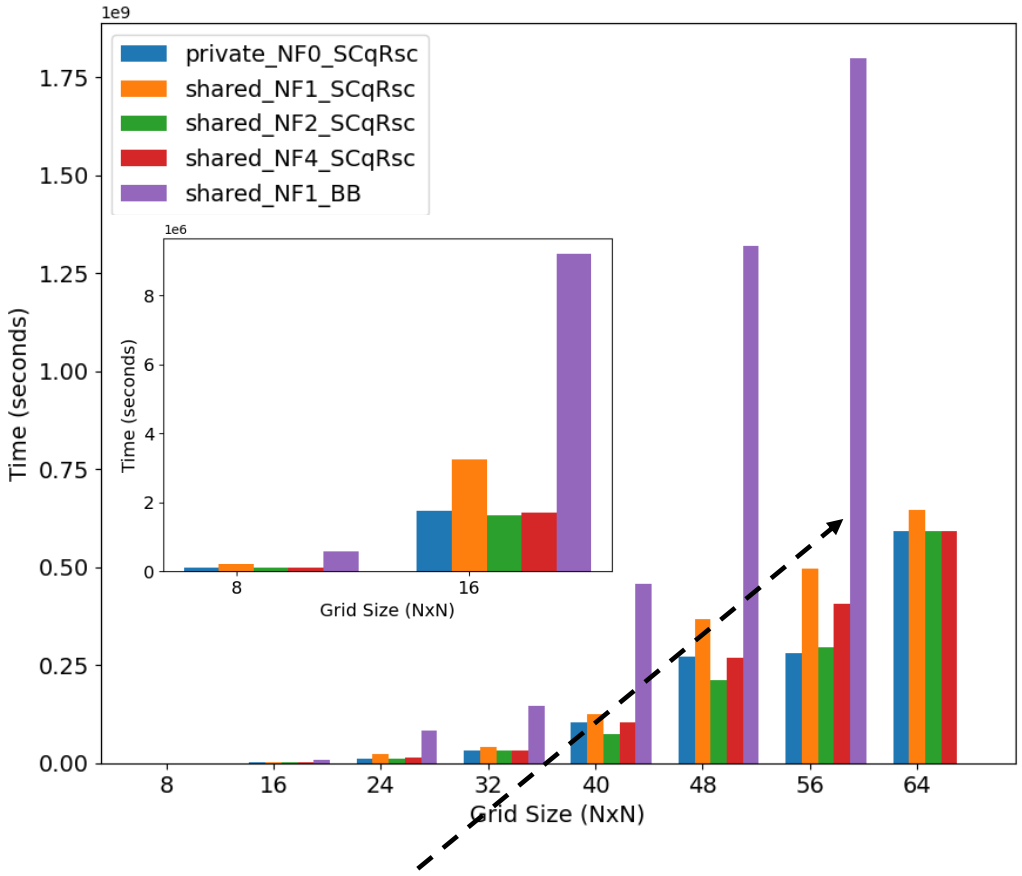
Grid with Shared Factories



Superconducting Qubit		IBM Bicycle	
Architecture Parameter	**Value**	**Architecture Parameter**	**Value**
Code type	Rotated surface	Code type	Gross/Two-Gross
Cycle time	1μs	Cycle time	1μs
Physical error	10^{-4}	Physical error	10^{-4}
Architecture type(s)	All	Architecture type(s)	Linear (shared factories)
Qubits per logical block	$2*d^2$	Qubits per logical block	400, 768
Intrablock gate limit	unlimited	Intrablock gate limit	1
Inter-block gate limit	1 per free path	Inter-block gate limit	1
Clifford time	d * 1μs	Clifford time	0
T-gate time	d * 1μs	T-gate time	324μs, 617μs
Magic-state factory	Optimized Litinski	Magic-state factory	Optimized Litinski

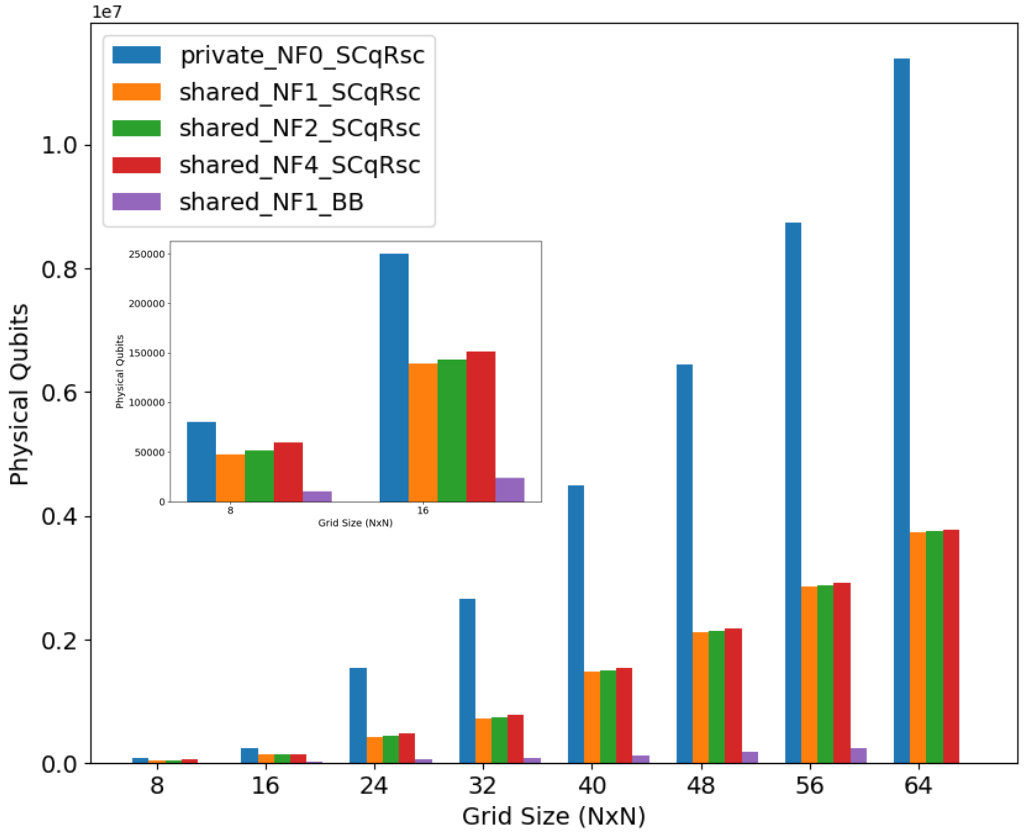
Runtime and Qubits for Heisenberg Dynamics

Runtime for one run of dynamics circuit



Problem size limited by achievable error rate

Physical qubits for one run of dynamics circuit

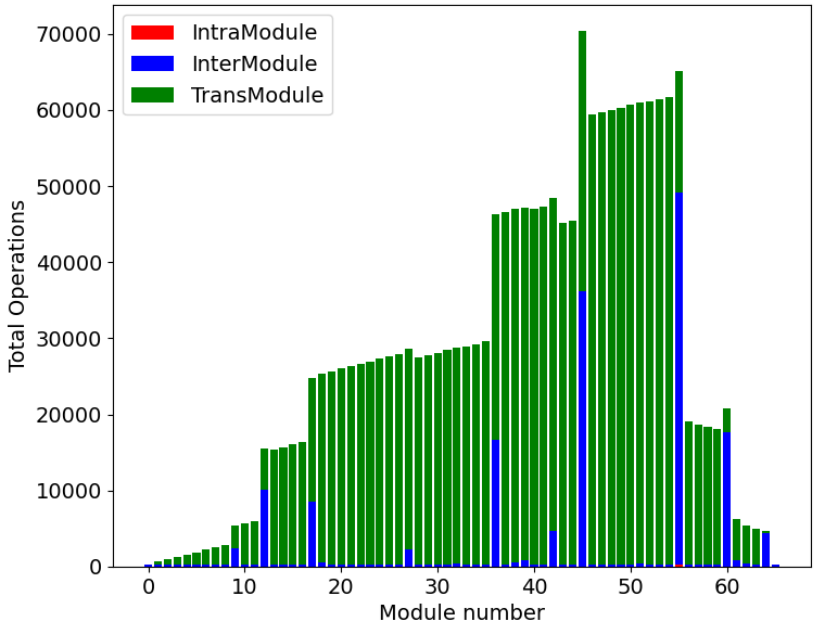


Physical error of 10^{-4}

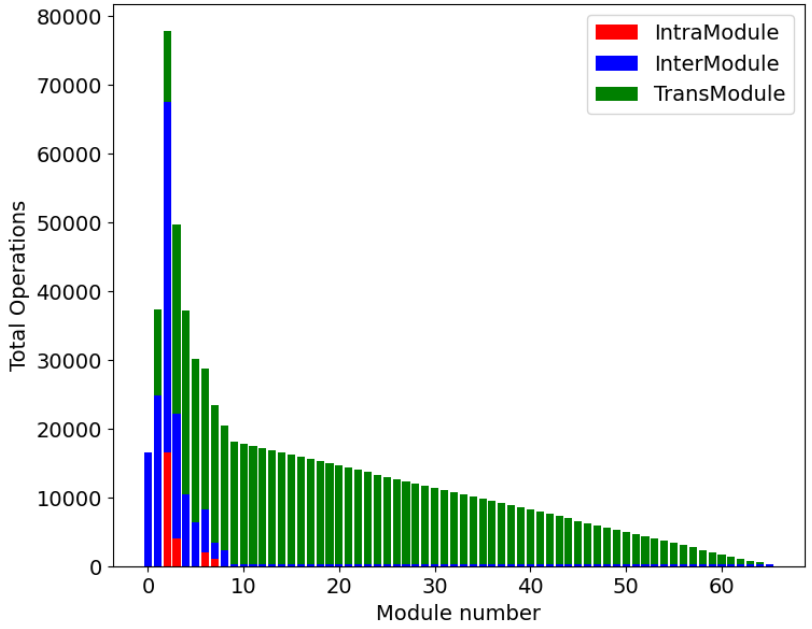
t_{max} = 1000
eps = 10^{-3}

Impact of Placement on Inter-block Operations

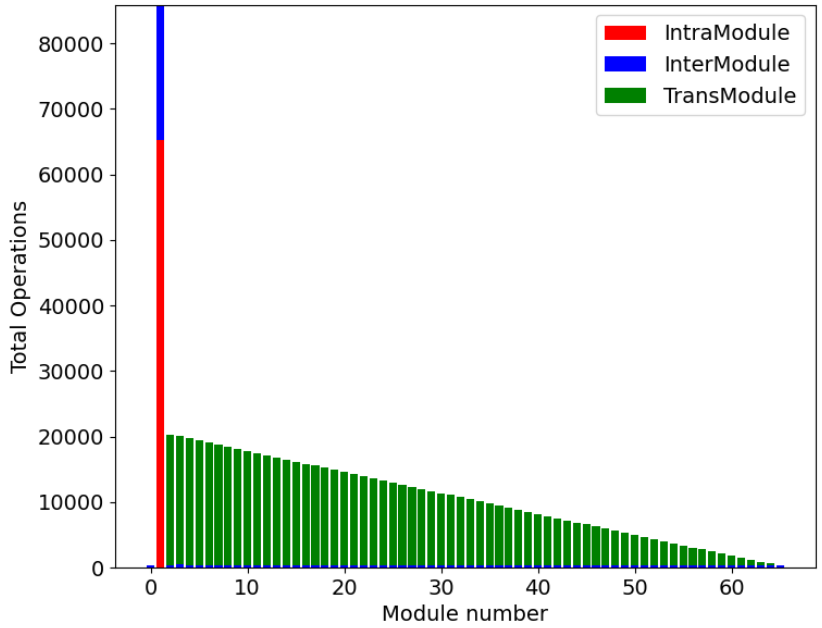
Random Placement of Qubits to Blocks



Balanced Placement of Qubits to Blocks



Localized Placement of Qubits to Blocks



Single block encoding of Heisenberg N=32

Module size: 16 qubits
64 total modules

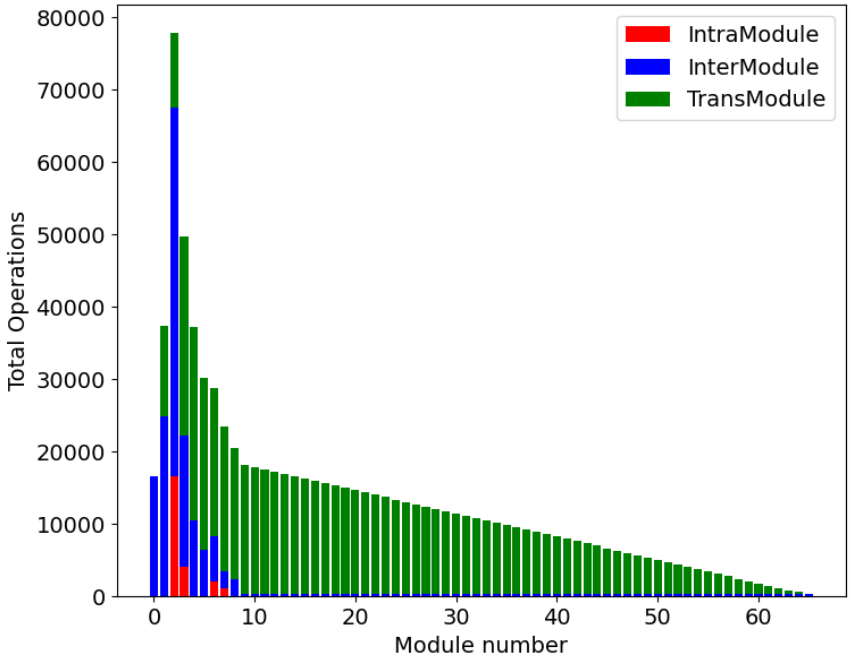
Types of operations:

- **IntraModule: two-qubit gates within module**
- **InterModule: two-qubit gates between modules**
- **TransModule: Operations that transit past modules**

2D Grid Layout of Modules

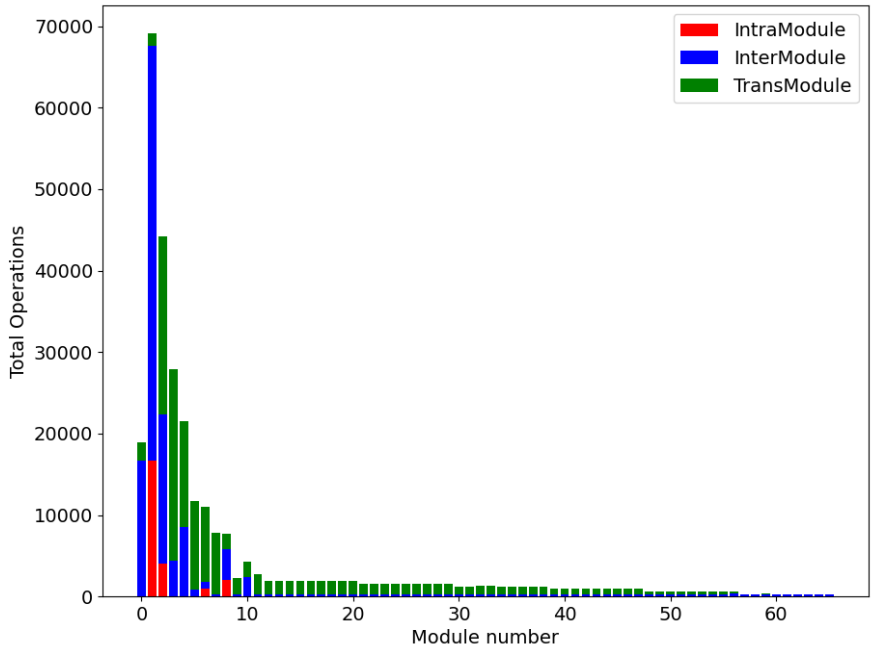
Linear Layout

Balanced Placement of Qubits to Blocks



2D Grid Layout

Balanced Placement of Qubits to Blocks



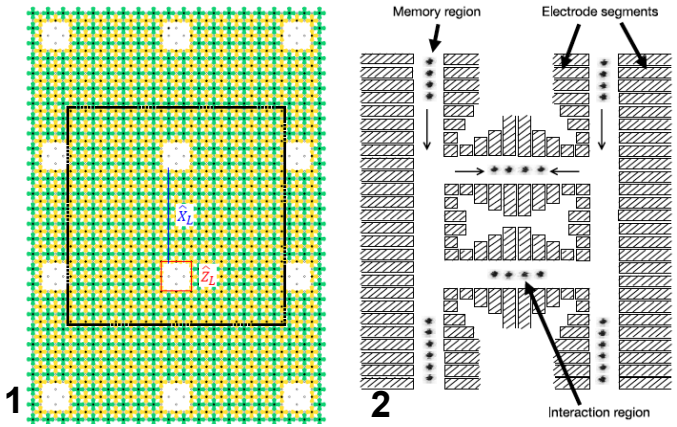
Single block encoding of Heisenberg N=32

Module size: 16 qubits
64 total modules

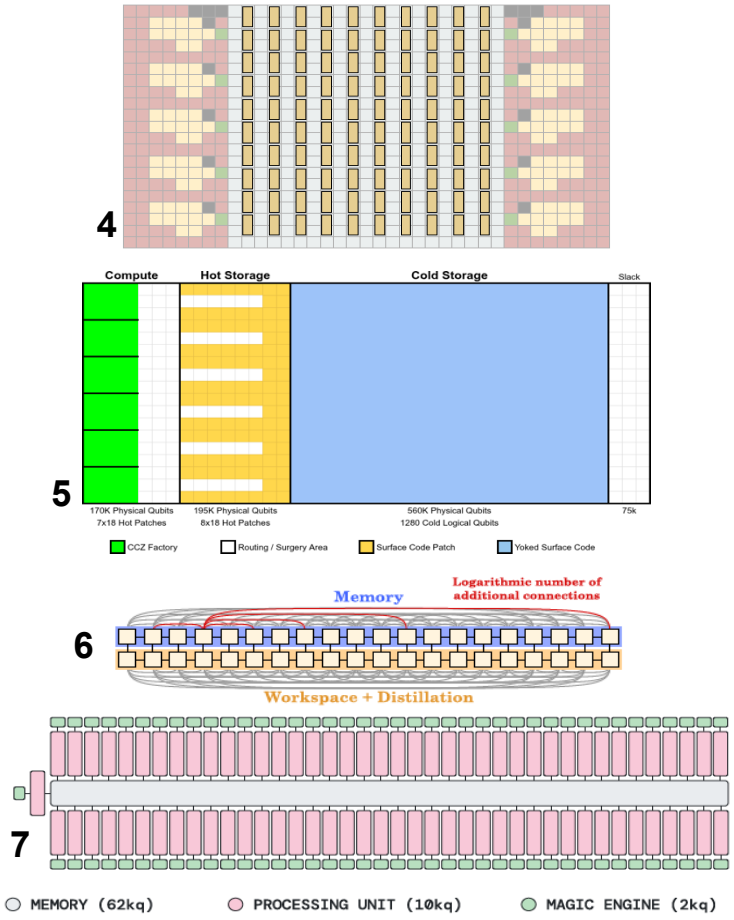
Types of operations:

- **IntraModule: two-qubit gates within module**
- **InterModule: two-qubit gates between modules**
- **TransModule: Operations that transit past modules**

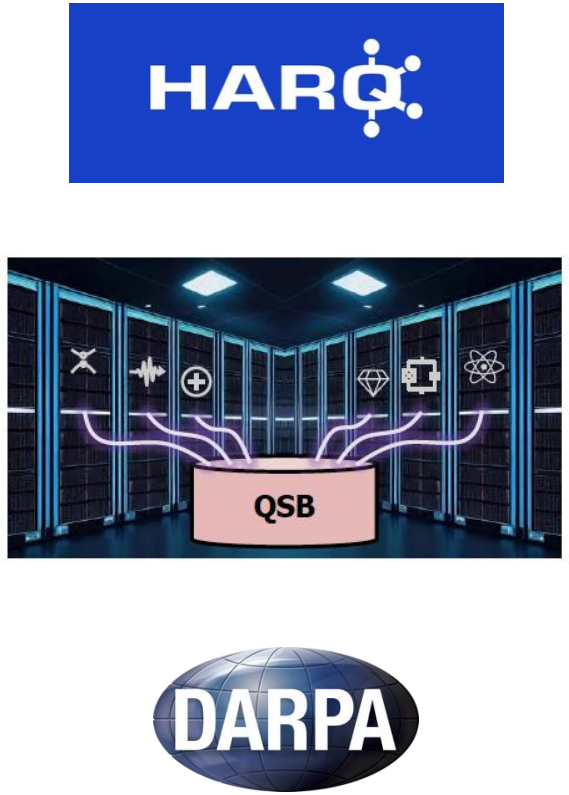
Challenges and Opportunities Ahead



Homogeneous technology and uniform design



Homogeneous technology and structured design



Heterogeneous technology and structured design

1) Fowler, et.al. “Surface codes: Toward practical large-scale QC.” Phys.Rev.A 86, 03324. (2012).
2) Kielpinski, et.al. “Architecture for a large-scale ion-trap QC.” Nature 417 (2002)
3) Veldhorst, et.al. *“Silicon CMOS architecture for a spin-based QC.” Nature Comm. 8, #1766 (2017).*
4) Litinski. “A Game of Surface Codes: Large-Scale QC with Lattice Surgery.” Quantum 3, 128 (2019)
5) Gidney. “How to factor 2048 RSA integers with less than a million noisy qubits.” arXiv: 2505.15917 (2025).
6) Caesura, et.al. “Faster quantum chemistry simulations on a QC with improved tensor factorization …” PRX Quantum 6, 030337. (2025)
7) Webster, et.al. “The Pinnacle Architecture.” arXiv:2602.11457

Summary/Conclusions

- **Resource estimation requires multiple steps:**
  - **Application specification, algorithm selection, circuit generation, resource counting**
- **Circuit-level analysis of architectures will have an increasing impact on fault-tolerant architecture develop in the future**

pyLIQTR:
pip install pyliqtr or
https://github.com/isi-usc-edu/pyLIQTR

Lincoln laboratory is hiring:
- QEC experts
- Algorithm design, optimization and analysis

Thanks to the Lincoln Laboratory quantum algorithms team!

Justin Elenewski

Kaitlyn Morrell

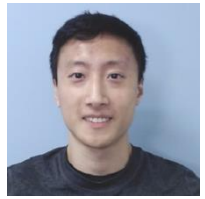
Parker Kuklinski

Ben Rempfer

Arthur Kurlej

Rylee Neumann

Rob Rood

Daniel Liang

Maria Prado Rodriguez