Quantum AI

Expressing and analyzing quantum algorithms with QUALTRAN

Presented by: Matt Harrigan

IWQC Workshop

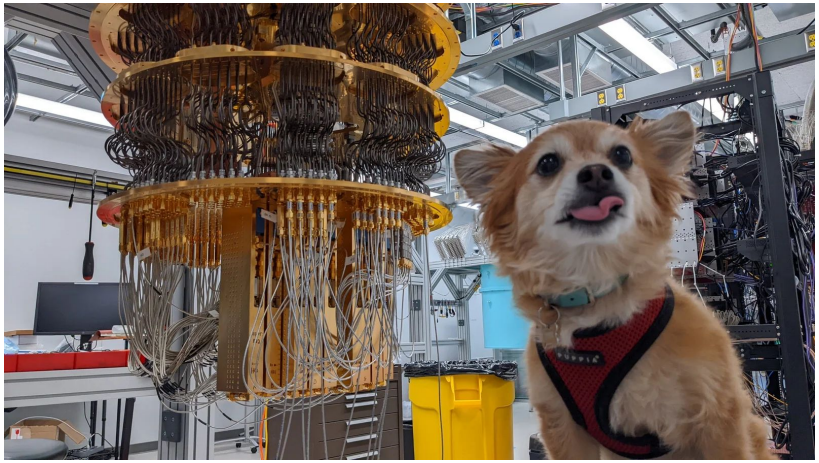
May 2023

Google

Why

Google is building error-corrected quantum computers

We want to know what we'll run and when. So should you!



Researchers are puzzling through the details:

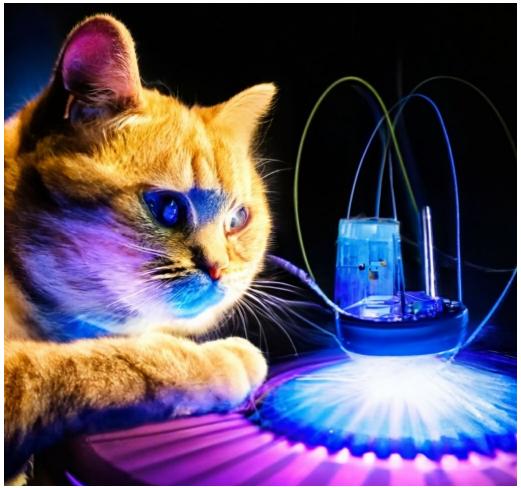
- [1905.09749] How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits
- [2202.01244] Reliably assessing the electronic structure of cytochrome P450 on today's classical computers and tomorrow's quantum computers
- [2302.05531] Fault-tolerant quantum simulation of materials using Bloch orbitals

Why (cont'd)

- [1905.09749] How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits
- [2202.01244] Reliably assessing the electronic structure of cytochrome P450 on today's classical computers and tomorrow's quantum computers
- [2302.05531] Fault-tolerant quantum simulation of materials using Bloch orbitals

It's **tedious**! Tooling and software will let us:

- Reduce toil and error
- Re-use common of algo primitives
- Demo algos in a more accessible way with visualization, code, and examples.



Expressing and Analyzing algorithms

We **cannot** currently run the algorithms we write down

We **can** make meaningful statements about their costs, composition, correctness, …

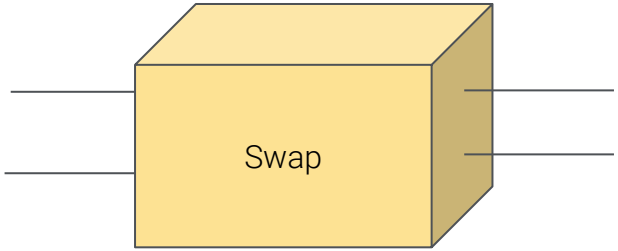
The bargain: the more you write down, the more you get out

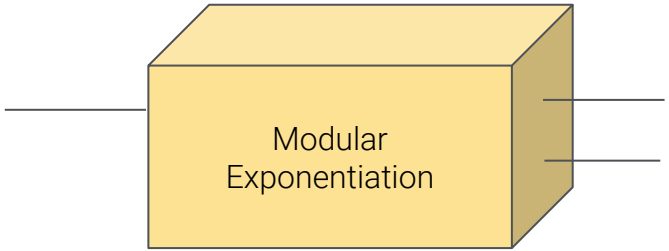
The corollary: you don't need to write down everything to get started

Reminder: Cirq-FT

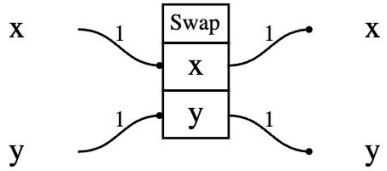
- Arithmetic Gates
  - AdditionaGate, AddMod, ContiguousRegisterGate, LessThanGate etc.
- State Preparation
  - PrepareUniformSuperposition: using a single round of amplitude amplification.
  - StatePreparationAliasSampling: QROM based state prep using classical alias sampling.
- Data Loading
  - QROM: Unary iteration based data loading using O(iteration_length) T-gates.
  - SelectSwapQROM: “advanced” QROM using O(sqrt(iteration_length)) T-gates.
- Qubitization
  - QubitizationWalkOperator, ReflectionUsingPrepare, SelectOracle, PrepareOracle
- Robin’s Mean Estimation Algorithm
  - MeanEstimationOperator, ComplexPhaseOracle, ArcTan etc.
- Others
  - UnaryIteration base class to enable expressing nested coherent for-loops using multi-dimensional selection registers.
  - ProgrammableRotationGateArray: QROM based rotation synthesis introduced in Guang Hao’s double factorization paper

Expressing algorithms with Bloqs

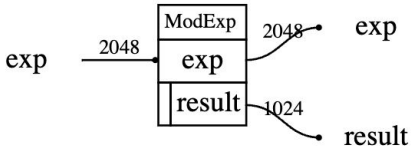





```python
class Swap(Bloq):
    @property
    def signature(self) -> 'Signature':
        return Signature.build(x=1, y=1)


swap = Swap()
show_bloq(swap)
```




```python
@dataclass(frozen=True)
class ModExp(Bloq):
    n: int

    @property
    def signature(self) -> 'Signature':
        return Signature([
            Register('exp', bitsize=2*self.n),
            Register('result', bitsize=self.n, side=Side.RIGHT)
        ])

mod_exp = ModExp(n=1024)
show_bloq(mod_exp)
```



Quantum AI

https://qualtran.readthedocs.io/en/latest/_infra/Bloqs-Tutorial.html

Bloqs are built out of other bloqs

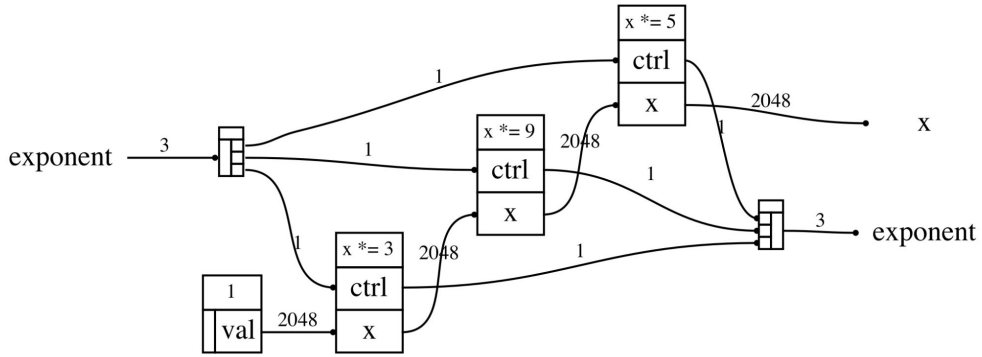
```python
def build_composite_bloq(self, bb: BloqBuilder, exponent):
    x = bb.add(IntState(val=1, bitsize=self.x_bitsize))
    exponent = bb.split(exponent)

    for j in range(self.exp_bitsize - 1, 0 - 1, -1):
        exponent[j], x = bb.add(CtrlModMul(k=self.base),
                                ctrl=exponent[j], x=x)
        base = base * base % self.mod

    return {'exponent': bb.join(exponent), 'x': x}
```

Define your bloq's implementation in terms of smaller bloqs.

Quantum variables follow linear logic

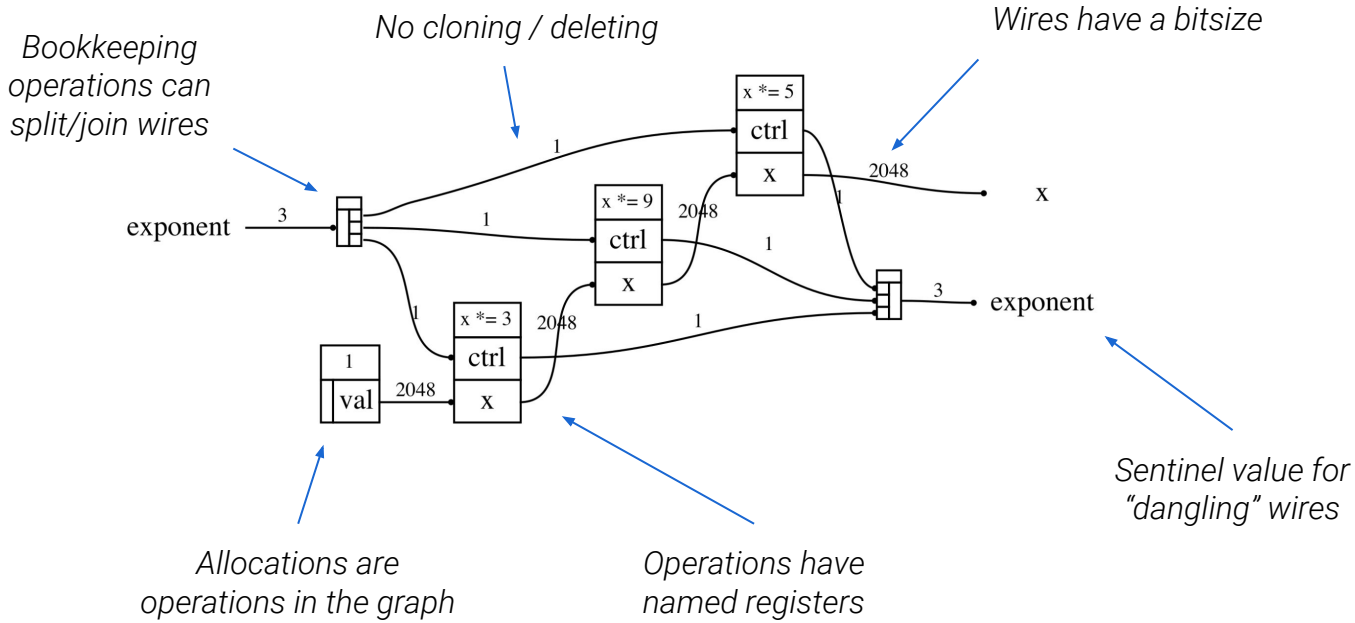
Our container type is `CompositeBloq`, which implements the `Bloq` interface

It is a directed, acyclic graph



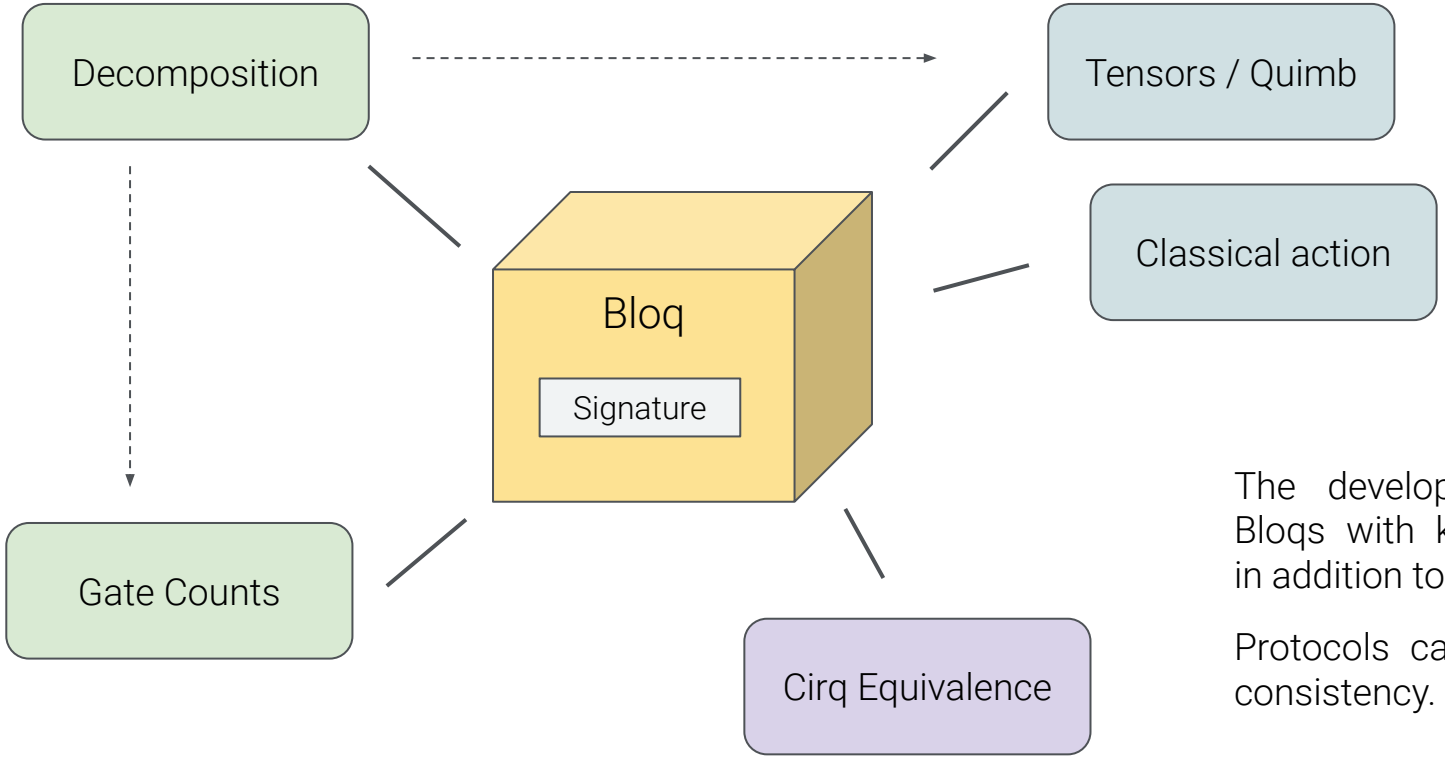
Quantum AI

Features of CompositeBloq



Quantum AI

Analyzing algorithms



The developer can annotate Bloqs with known information in addition to its decomposition

Protocols can be checked for consistency.

Classical Reversible Simulation

```python
class ModExp(Bloq):

    ...

    def on_classical_vals(self, exponent: int):
        return {'exponent': exponent,
                'x': (self.base ** exponent) % self.mod}
```

```python
class CtrlModMul(Bloq):

    ...

    def on_classical_vals(self, ctrl: int, x: int):
        if ctrl == 0:
            return {'ctrl': ctrl, 'x': x}

        return {'ctrl': ctrl,
                'x': (x * self.k) % self.mod}
```

Annotate with classical action (where appropriate)

```python
N = 13*17
n = int(np.ceil(np.log2(N)))
g = 8

mod_exp = ModExp(base=g, mod=N, exp_bitsize=32, x_bitsize=32)
mod_exp_decomp = mod_exp.decompose_bloq()

for e in range(20):
    ref = (g ** e) % N
    _, bloq_eval = mod_exp.call_classically(exponent=e)
    _, decomp_eval = mod_exp_decomp.call_classically(exponent=e)

    print(f'{e:5d}: {ref:5d} {bloq_eval:5d} {decomp_eval:5d}')
```

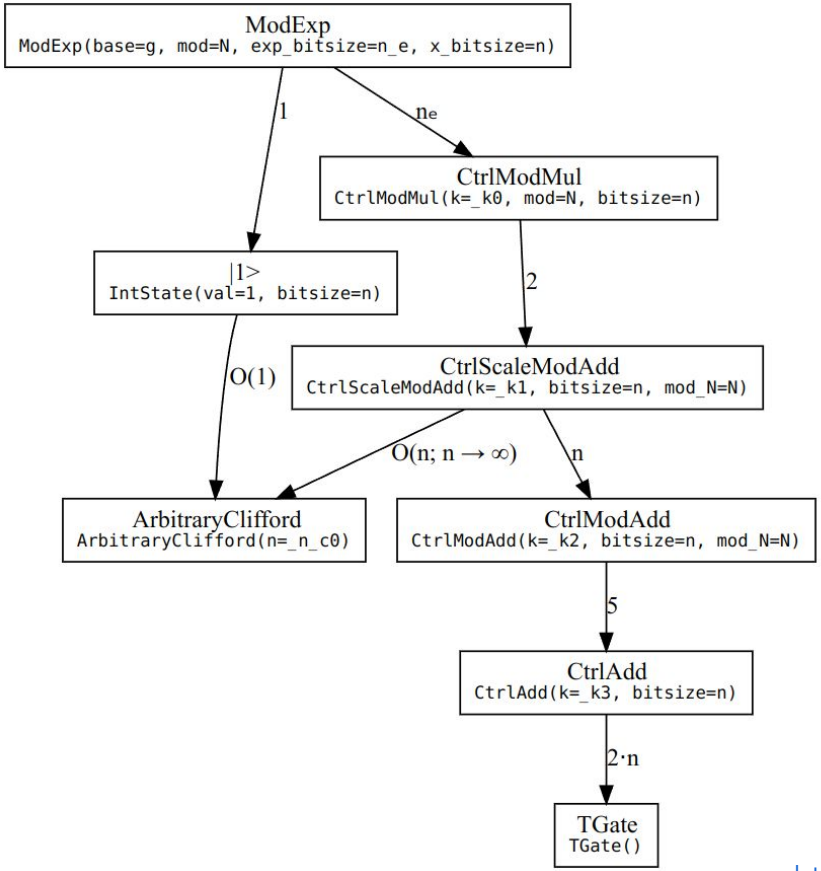
```
    0:     1     1     1
    1:     8     8     8
    2:    64    64    64
    3:    70    70    70
    4:   118   118   118
    5:    60    60    60
    6:    38    38    38
    7:    83    83    83
    8:     1     1     1
    9:     8     8     8
   10:    64    64    64
   11:    70    70    70
   12:   118   118   118
   13:    60    60    60
   14:    38    38    38
   15:    83    83    83
```

Fuzz test your decomposition on classical inputs

Quantum AI

https://qualtran.readthedocs.io/en/latest/simulation/classical_sim.html

Symbolics and Gate Counting



Annotate bloqs with expressions for (sub-)bloq counts

Big-O the parts you don't care about

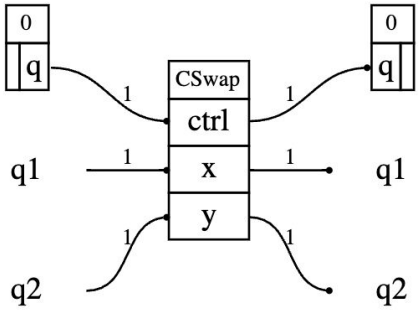
More consistency checks

Visualize your decomposition hierarchy

https://qualtran.readthedocs.io/en/latest/bloqs/factoring/ref-factoring.html

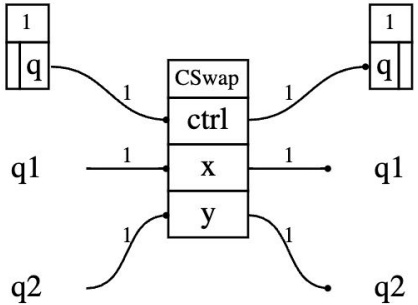
Numerical Simulation and Tensor Contraction

```
ctrl = bb.add(ZeroState())
ctrl, q1, q2 = bb.add(CSwap(), ctrl=ctrl, x=q1, y=q2)
bb.add(ZeroEffect(), q=ctrl)
deactivated_cswap = bb.finalize(q1=q1, q2=q2)

show_bloq(deactivated_cswap)
deactivated_cswap.tensor_contract().real
```




```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

```
ctrl = bb.add(OneState())
ctrl, q1, q2 = bb.add(CSwap(), ctrl=ctrl, x=q1, y=q2)
bb.add(OneEffect(), q=ctrl)
activated_cswap = bb.finalize(q1=q1, q2=q2)


show_bloq(activated_cswap)
activated_cswap.tensor_contract().real
```




```
array([[1., 0., 0., 0.],
       [0., 0., 1., 0.],
       [0., 1., 0., 0.],
       [0., 0., 0., 1.]])
```

Quantum AI

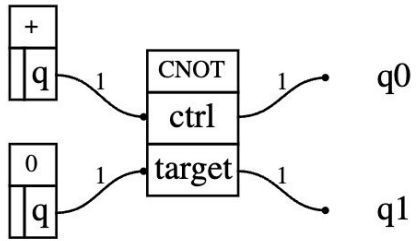
Bi-directional Cirq Interop

`CirqGateAsBloq` and `BloqAsCirqGate` both exist

As does conversion to and from `cirq.Circuit`

Subject to Cirq limitations: Shim to flat array of individual qubits.

```
bb = BloqBuilder()
q0 = bb.add(PlusState())
q1 = bb.add(ZeroState())

q0, q1 = bb.add(CNOT(), ctrl=q0, target=q1)

bell = bb.finalize(q0=q0, q1=q1)
show_bloq(bell)
```




```
circuit, qubits = bell.to_cirq_circuit()
circuit
```

```
_c(0): ──H──@──
            │
_c(1): ─────X──
```

Quantum AI

https://qualtran.readthedocs.io/en/latest/cirq_interop/cirq_interop.html

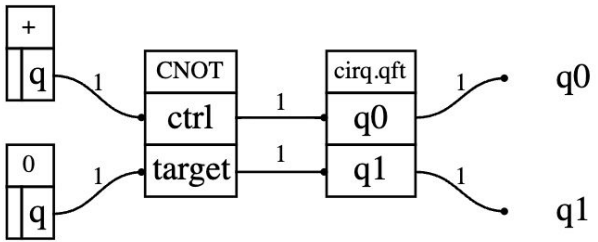
Bi-directional Cirq Interop

`CirqGateAsBloq` and `BloqAsCirqGate` both exist

As does conversion to and from `cirq.Circuit`

Subject to Cirq limitations: Shim to flat array of individual qubits.

```
qft = cirq.QuantumFourierTransformGate(num_qubits=2)
q0, q1 = bb.add(CirqGateAsBloq(qft), qubits=[q0,q1])

bell_qft = bb.finalize(q0=q0, q1=q1)
show_bloq(bell_qft)
```




```
circuit, qubits = bell_qft.to_cirq_circuit()
circuit
```

```
_c(0): ──H──@──qft──
            │  │
_c(1): ─────X──#2───
```

Quantum AI

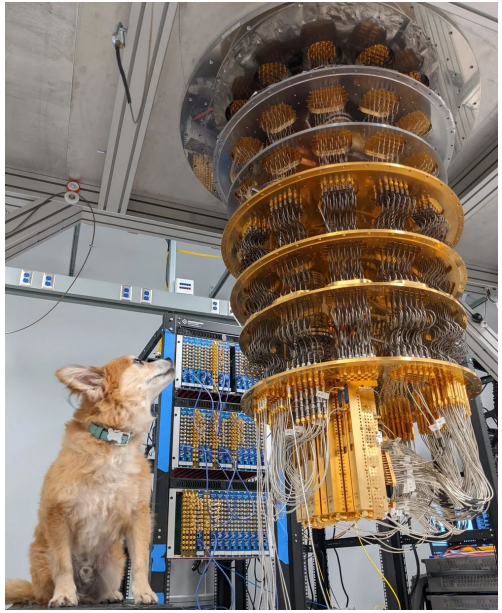
https://qualtran.readthedocs.io/en/latest/cirq_interop/cirq_interop.html

Qualtran Experimental Preview

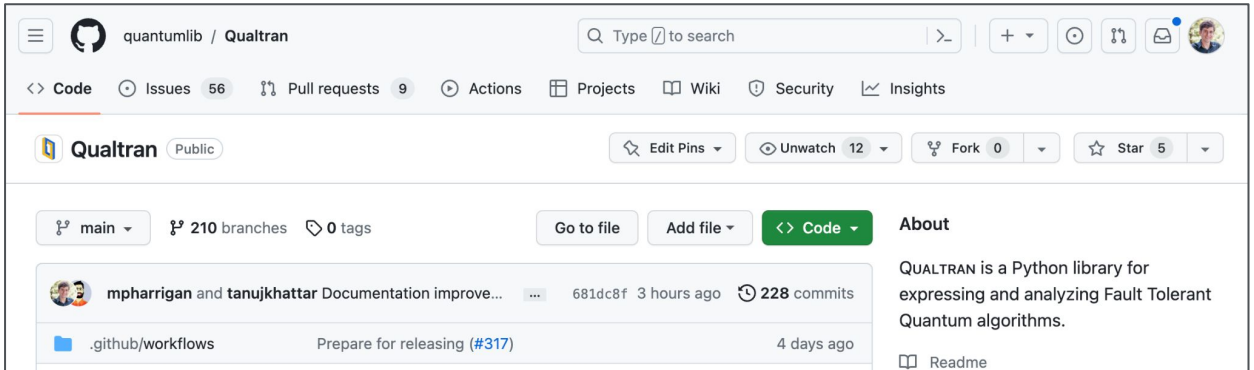
We know things about the algorithms. Let's write them down in a structured way!

Qualtran is open source as an experimental preview.
A lot more to come!

Please **get in touch** if you're interested in contributing!
mpharrigan@google.com
https://github.com/quantumlib/Qualtran



- *Raising issues…*
- *Writing Bloqs…*
- *Correctness protocols…*
- *Visualization protocols…*



Quantum AI

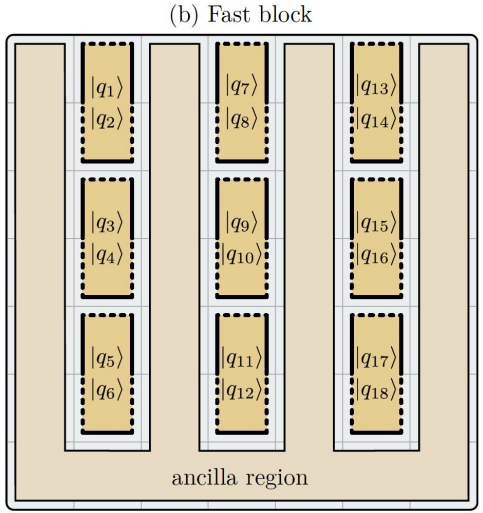
Physical costs

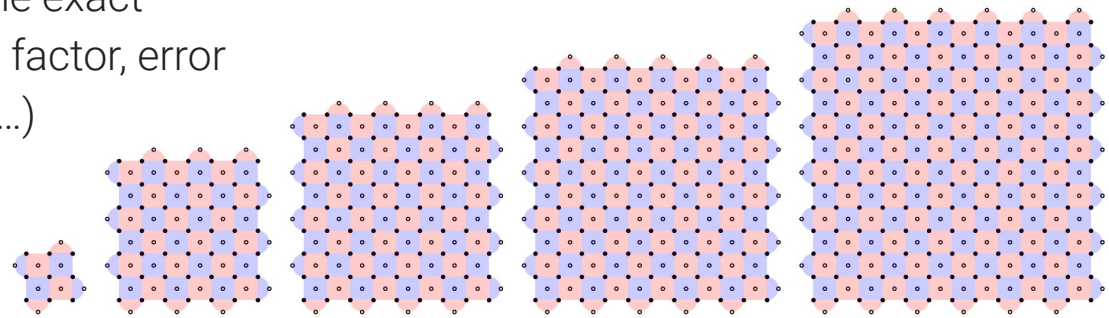
Google is targeting lattice surgery on the (rotated) surface code with T or CCZ factories with Lambda~10 and 1us cycle time.

“Game of surface codes” compilation is fully general and gives precise numbers.

Manual layout is very involved. Automated optimizing layout is a huge software challenge.

Still a lot of uncertainty around the exact architecture (shape, suppression factor, error tolerance, cosmic rays, timeline, …)





Quantum AI