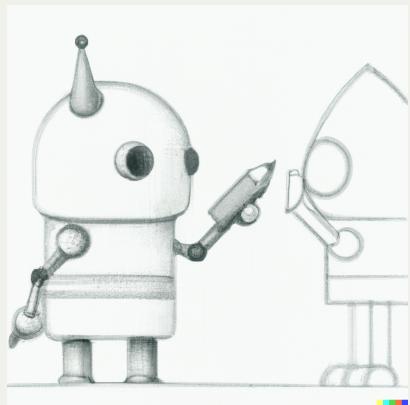


Hierarchical Quantum Circuit Representations



A cute robot building itself with artificial intelligence, pencil drawing - DALL-E 2





A cute robot building itself with artificial intelligence, pencil
drawing - DALL-E 3

Hierarchical Quantum Circuit Representations

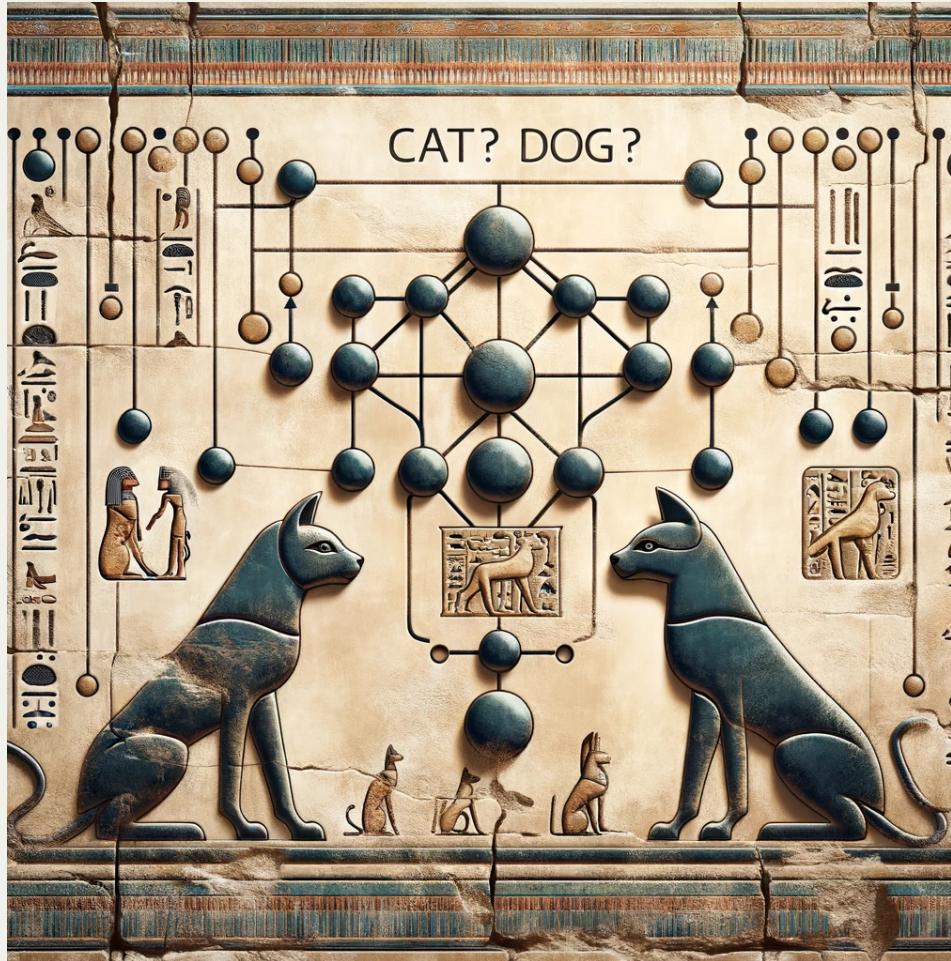


Background

Neural Networks

Images Generated with DALL-E 3

Neural Networks



Images Generated with DALL-E 3

AlexNet

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called “dropout” that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

AlexNet

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called “dropout” that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

Model	Top-1	Top-5
<i>Sparse coding [2]</i>	47.1%	28.2%
<i>SIFT + FVs [24]</i>	45.7%	25.7%
CNN	37.5%	17.0%

Table 1: Comparison of results on ILSVRC-2010 test set. In *italics* are best results achieved by others.

AlexNet

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called “dropout” that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

Model	Top-1	Top-5
<i>Sparse coding [2]</i>	47.1%	28.2%
<i>SIFT + FVs [24]</i>	45.7%	25.7%
CNN	37.5%	17.0%

Table 1: Comparison of results on ILSVRC-2010 test set. In *italics* are best results achieved by others.

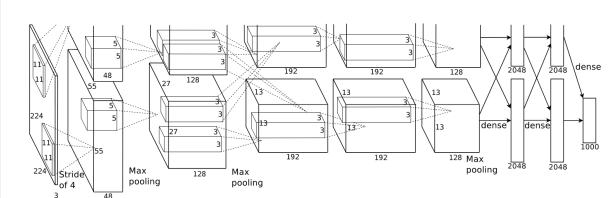
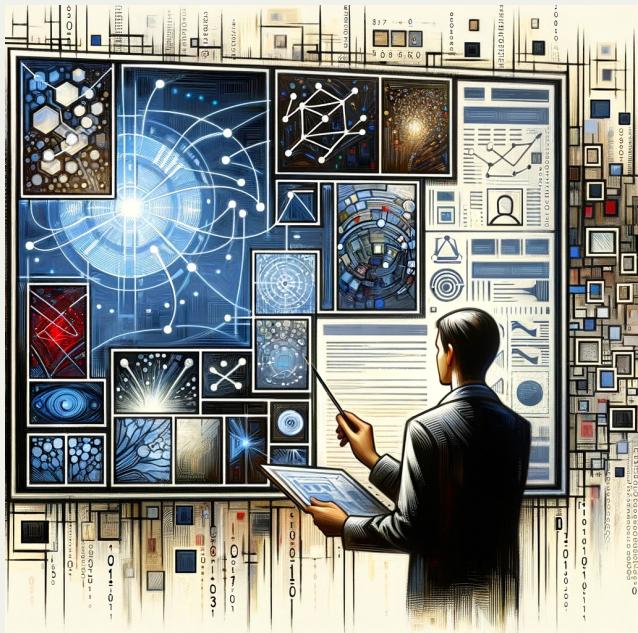


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network’s input is 150,528-dimensional, and the number of neurons in the network’s remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

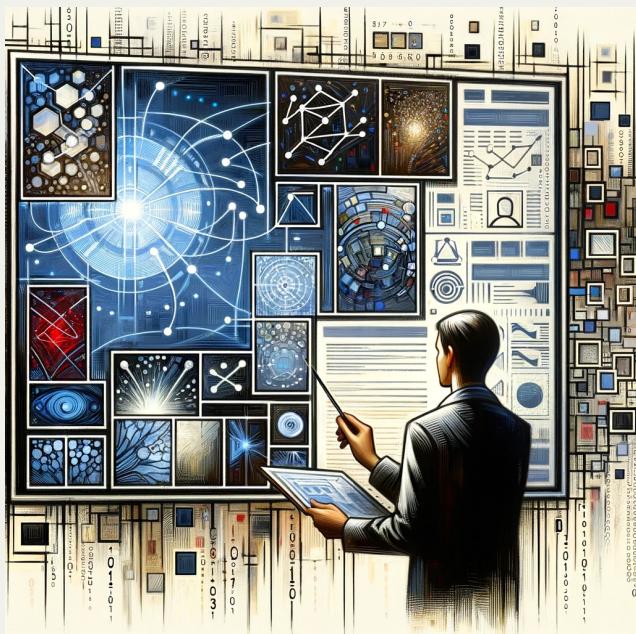
Automation



Images Generated with DALL-E 3

Automation

Manual



Automated

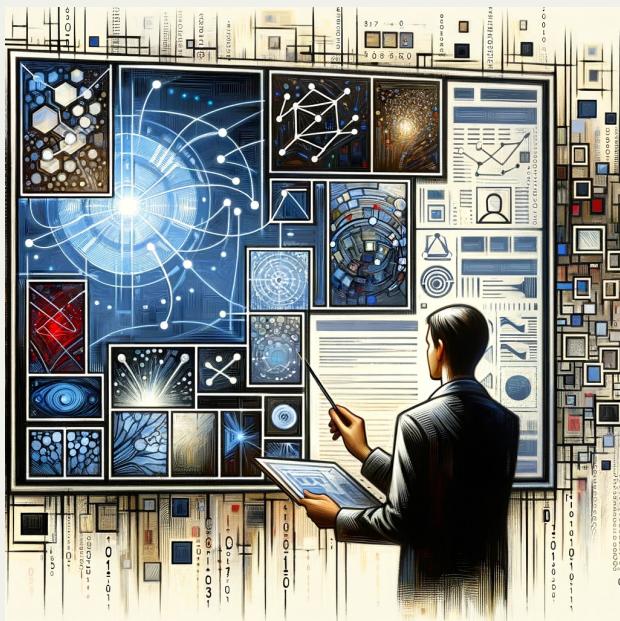


→
Feature Engineering

Images Generated with DALL-E 3

Automation

Manual



Automated

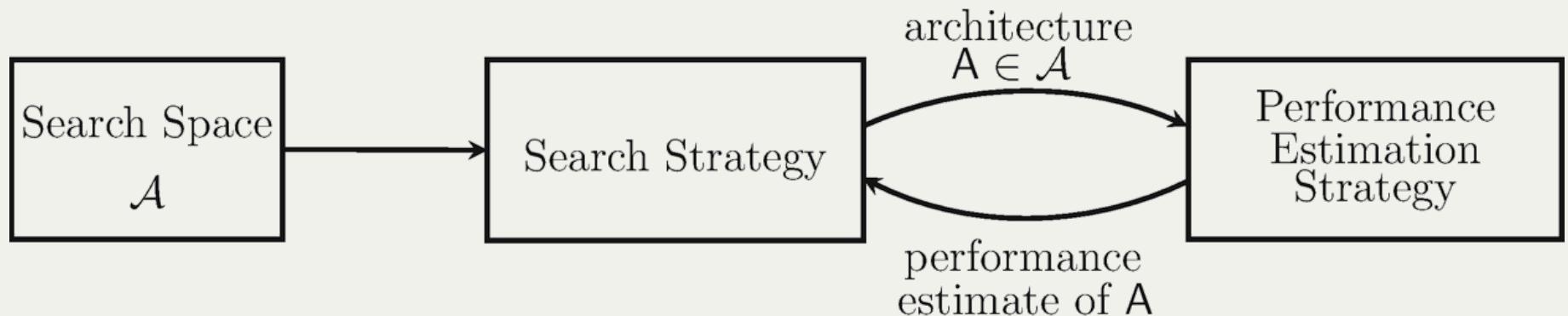


→
Architecture
Engineering?

Images Generated with DALL-E 3

Neural Architecture Search

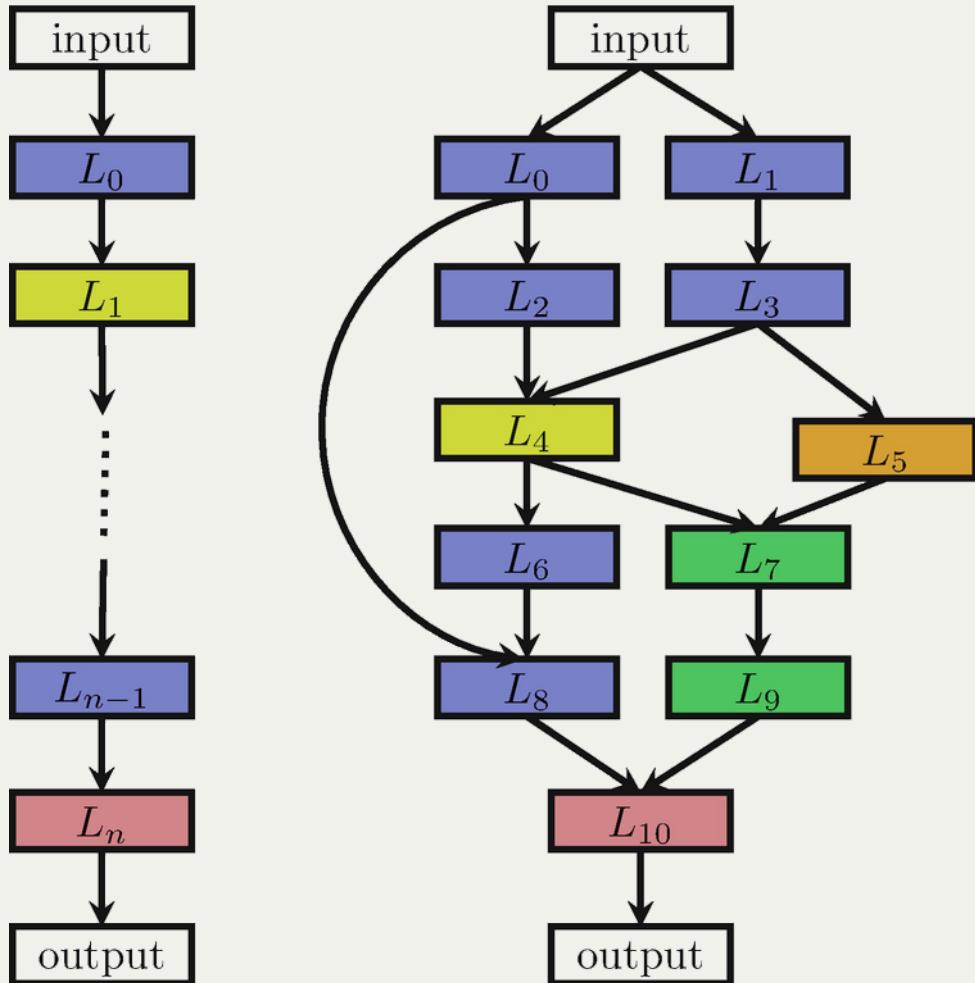
Neural Architecture Search



***Fig:** Abstract illustration of Neural Architecture Search methods. A search strategy selects an architecture A from a predefined search space \mathcal{A} . The architecture is passed to a performance estimation strategy, which returns the estimated performance of A to the search strategy.

*Elsken, T., Metzen, J. H. & Hutter, F. Neural architecture search: a survey. *J. Mach. Learn. Res.* 20, 1-21 (2019).

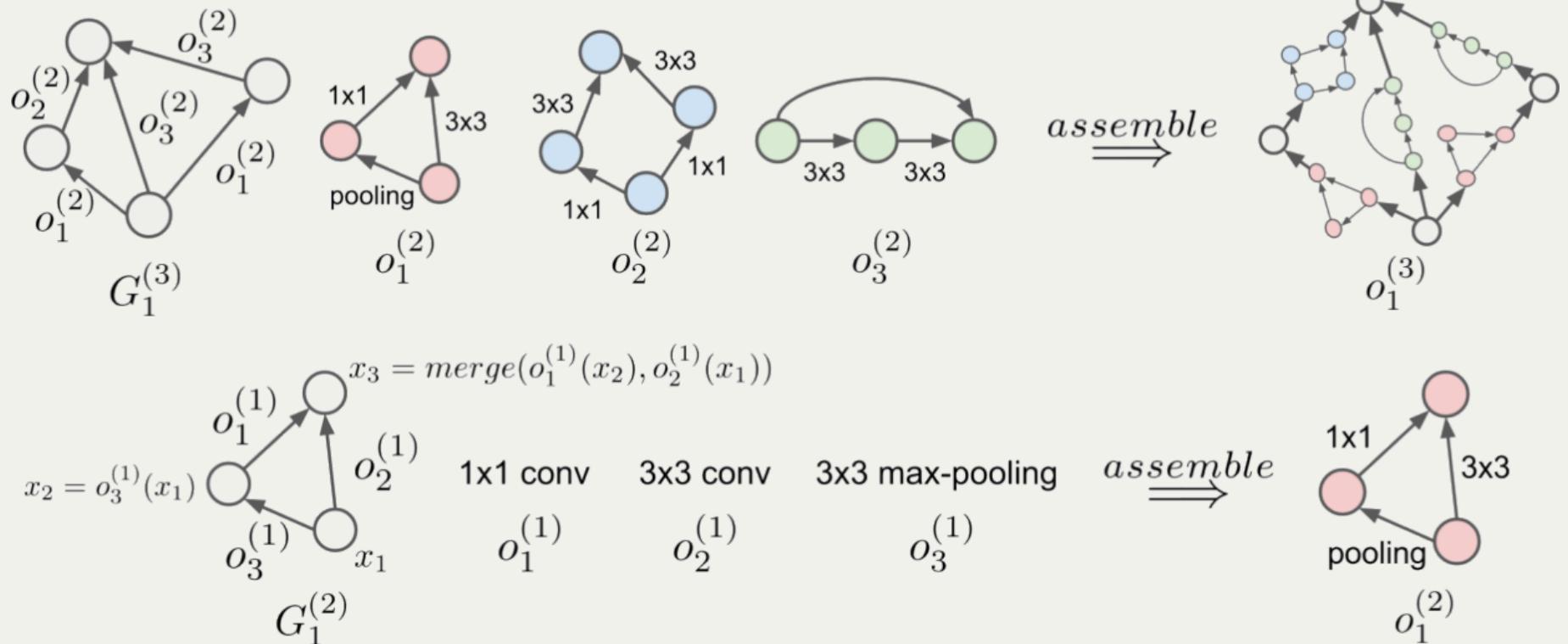
Neural Architecture Search



***Fig:** An illustration of different architecture spaces. Each node in the graphs corresponds to a layer in a neural network, e.g., a convolutional or pooling layer. Different layer types are visualized by different colors. An edge from layer L_i to layer L_j denotes that L_j receives the output of L_i as input. Left: an element of a chain-structured space. Right: an element of a more complex search space with additional layer types and multiple branches and skip connections.

*Elsken, T., Metzen, J. H. & Hutter, F. Neural architecture search: a survey. *J. Mach. Learn. Res.* 20, 1-21 (2019).

Neural Architecture Search

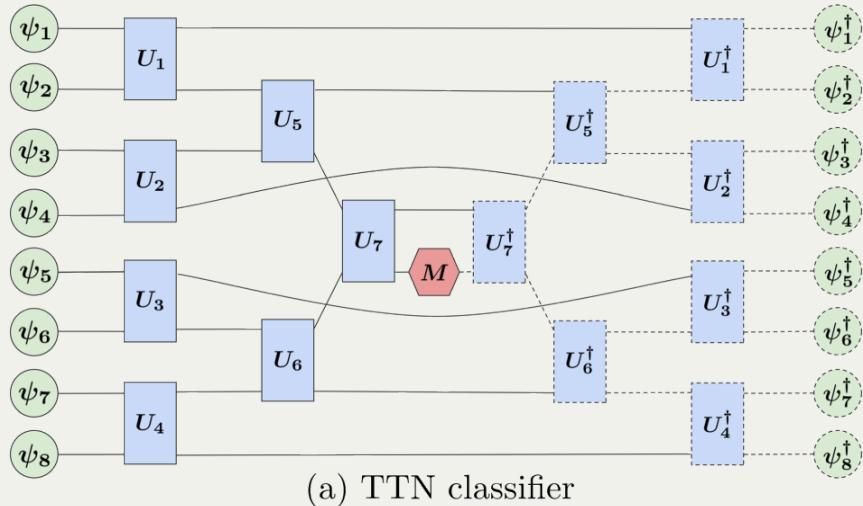


***Fig:** An example of a three-level hierarchical architecture representation. The bottom row shows how level-1 primitive operations $o_1^{(1)}$, $o_2^{(1)}$, $o_3^{(1)}$ are assembled into a level-2 motif $o_1^{(2)}$. The top row shows how level-2 motifs $o_1^{(2)}$, $o_2^{(2)}$, $o_3^{(2)}$ are then assembled into a level-3 motif $o_1^{(3)}$.

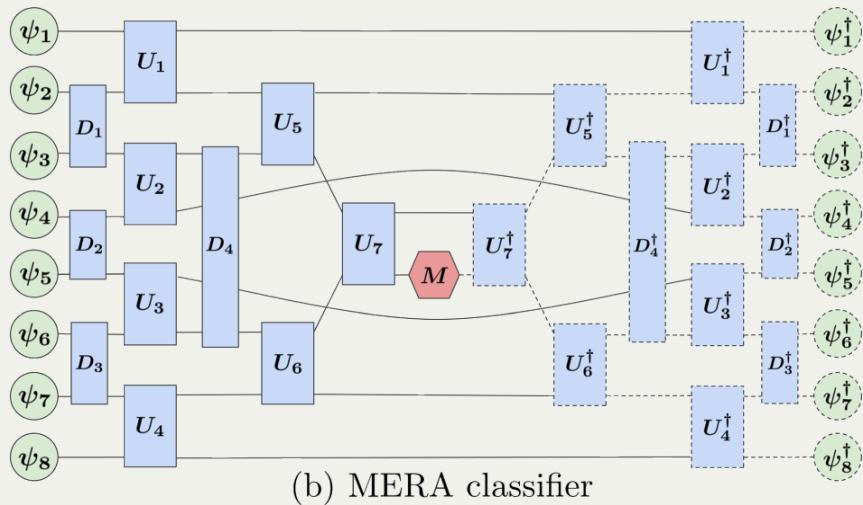
*Liu, H., Simonyan, K., Vinyals, O., Fernando, C. & Kavukcuoglu, K. Hierarchical representations for efficient architecture search. Int. Conf. Learn. Represent. (2018).

Ansatz Design

Ansatz Design



(a) TTN classifier

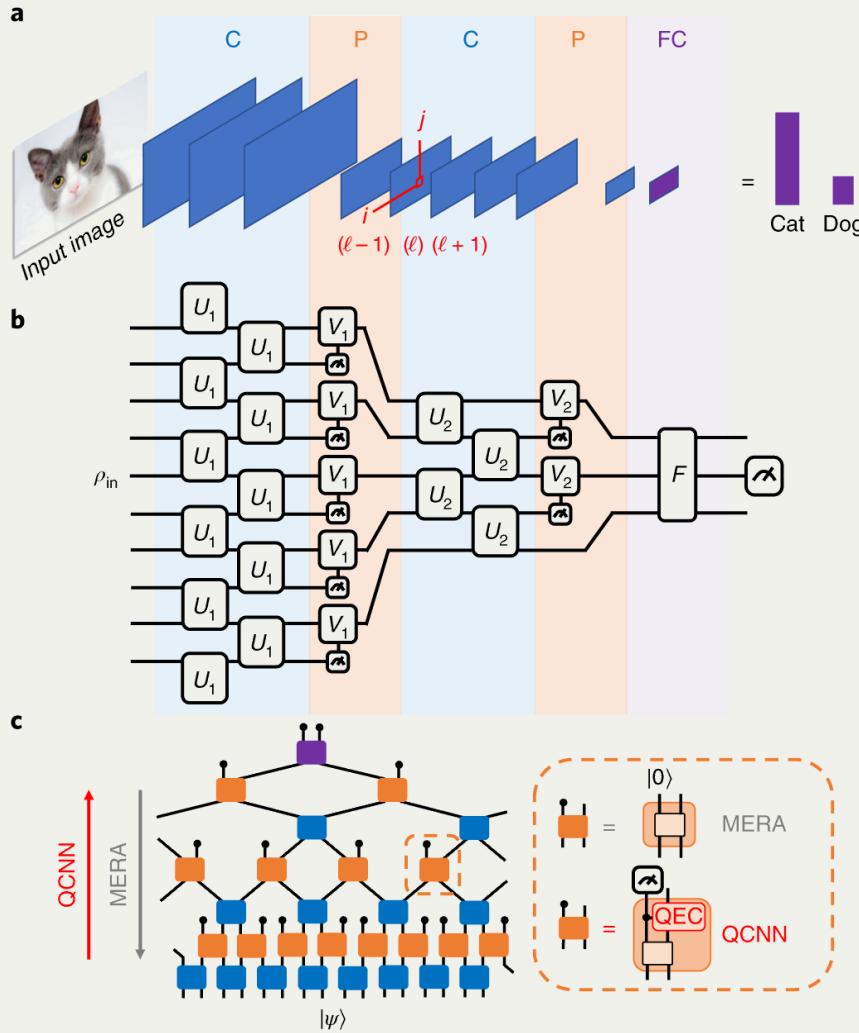


(b) MERA classifier

***Fig:** TTN and MERA classifiers for eight qubits. The quantum circuit is illustrated by the regions outlined in solid lines comprising inputs ψ , unitary blocks $\{U_i\}_{i=1}^7$ and $\{D_i\}_{i=1}^4$, and a measurement operator M . The dashed lines represent its conjugate transpose. The solid and dashed regions together describe a tensor network operating on input ψ_{1-8} and evaluating to the expectation value of observable M

*Grant, E. et al. Hierarchical quantum classifiers. NPJ Quantum Inf. 4, 65 (2018).

Ansatz Design



***Fig:** The concept of QCNNs. *a*, Simplified illustration of classical CNNs. A sequence of image-processing layers transforms an input image into a series of feature maps (blue rectangles) and finally into an output probability distribution (purple bars). C, convolution; P, pooling; FC, fully connected. *b*, QCNNs inherit a similar layered structure. Boxes represent unitary gates or measurement with feed-forwarding. *c*, The QCNN and the MerA share the same circuit structure, but run in reverse directions.

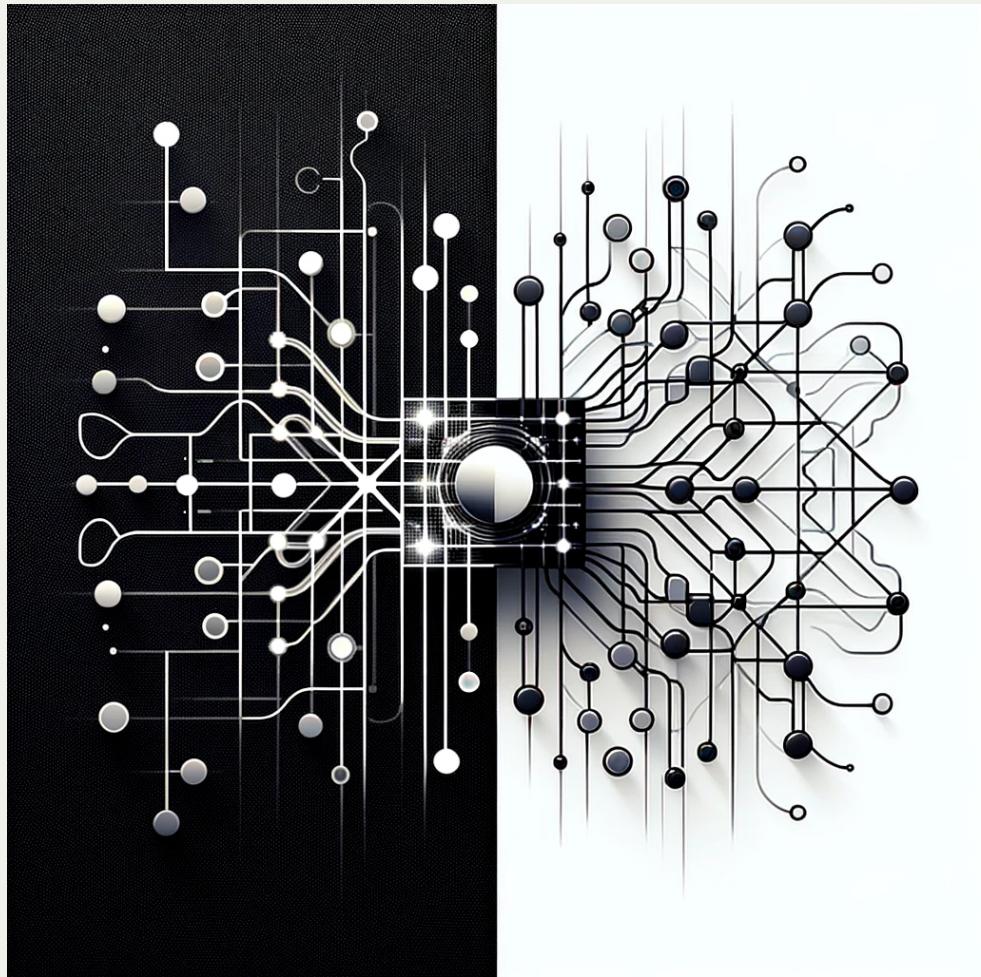
*Cong, I., Choi, S. & Lukin, M. D. Quantum convolutional neural networks. Nat. Phys. 15, 1273–1278 (2019). Image of cat: <https://www.pexels.com/photo/grey-and-white-short-furcat-104827/>.

Compute Graph Design

Program Synthesis

Compute Graph Design

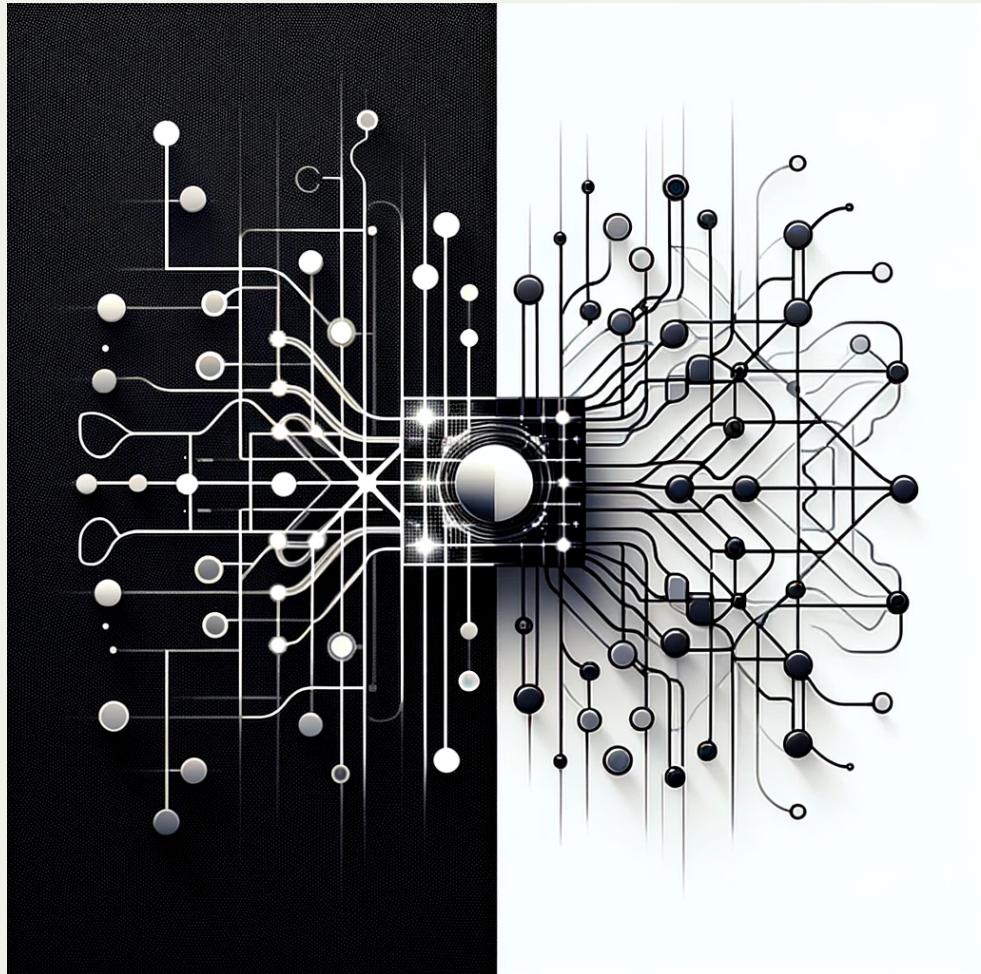
Program Synthesis



Compute Graph Design

Program Synthesis

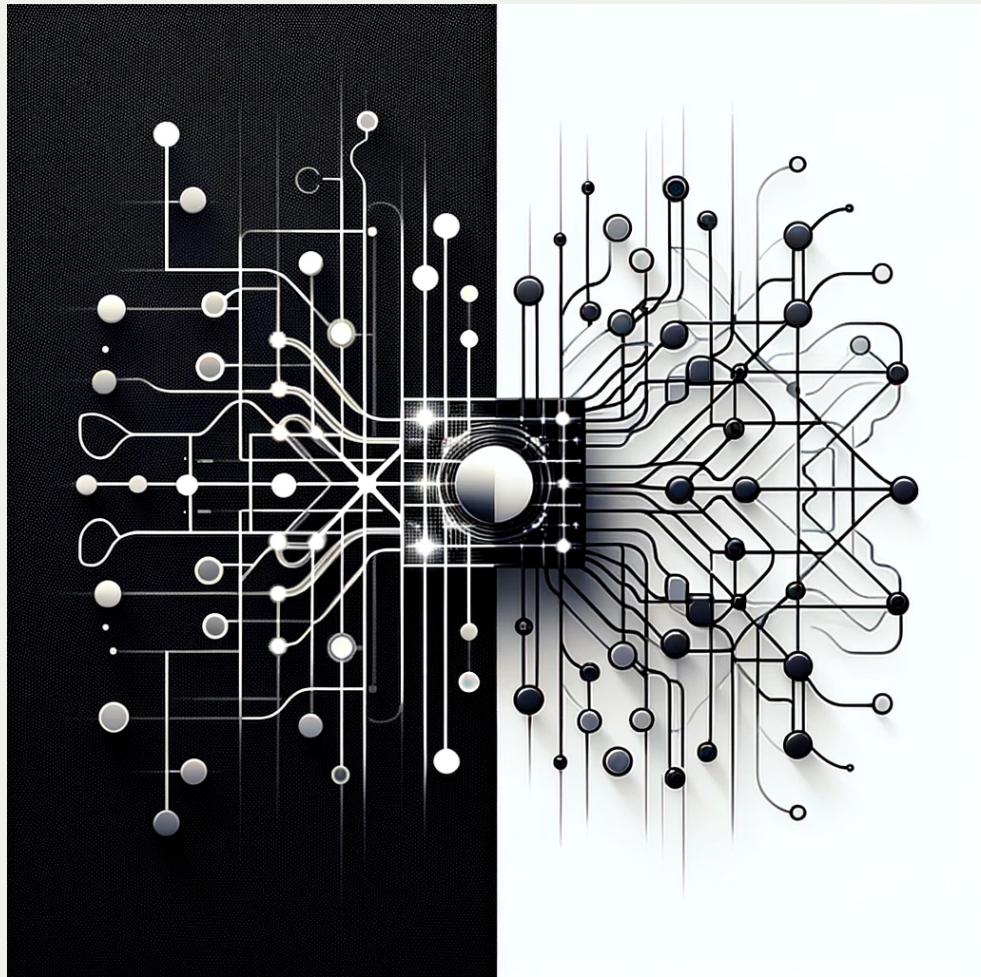
- Neural Networks



Compute Graph Design

Program Synthesis

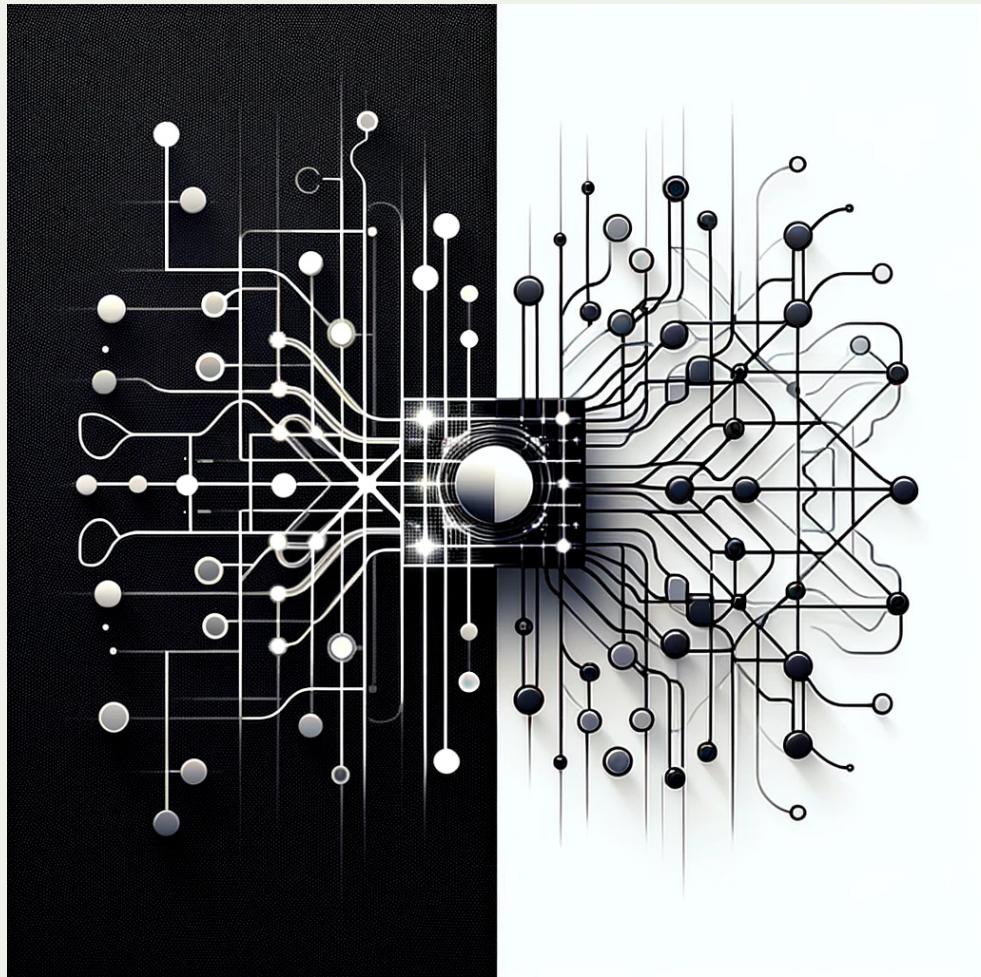
- Neural Networks
- Tensor Networks



Compute Graph Design

Program Synthesis

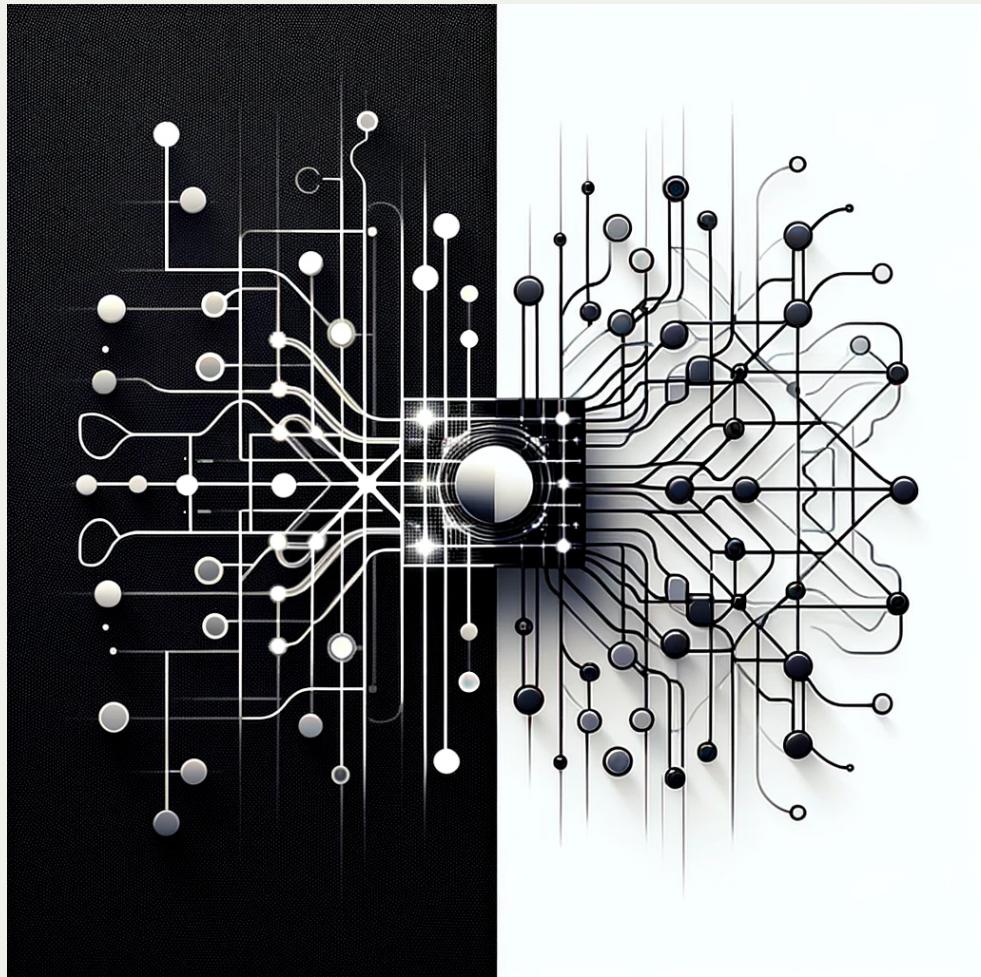
- Neural Networks
- Tensor Networks
- Quantum Circuits



Compute Graph Design

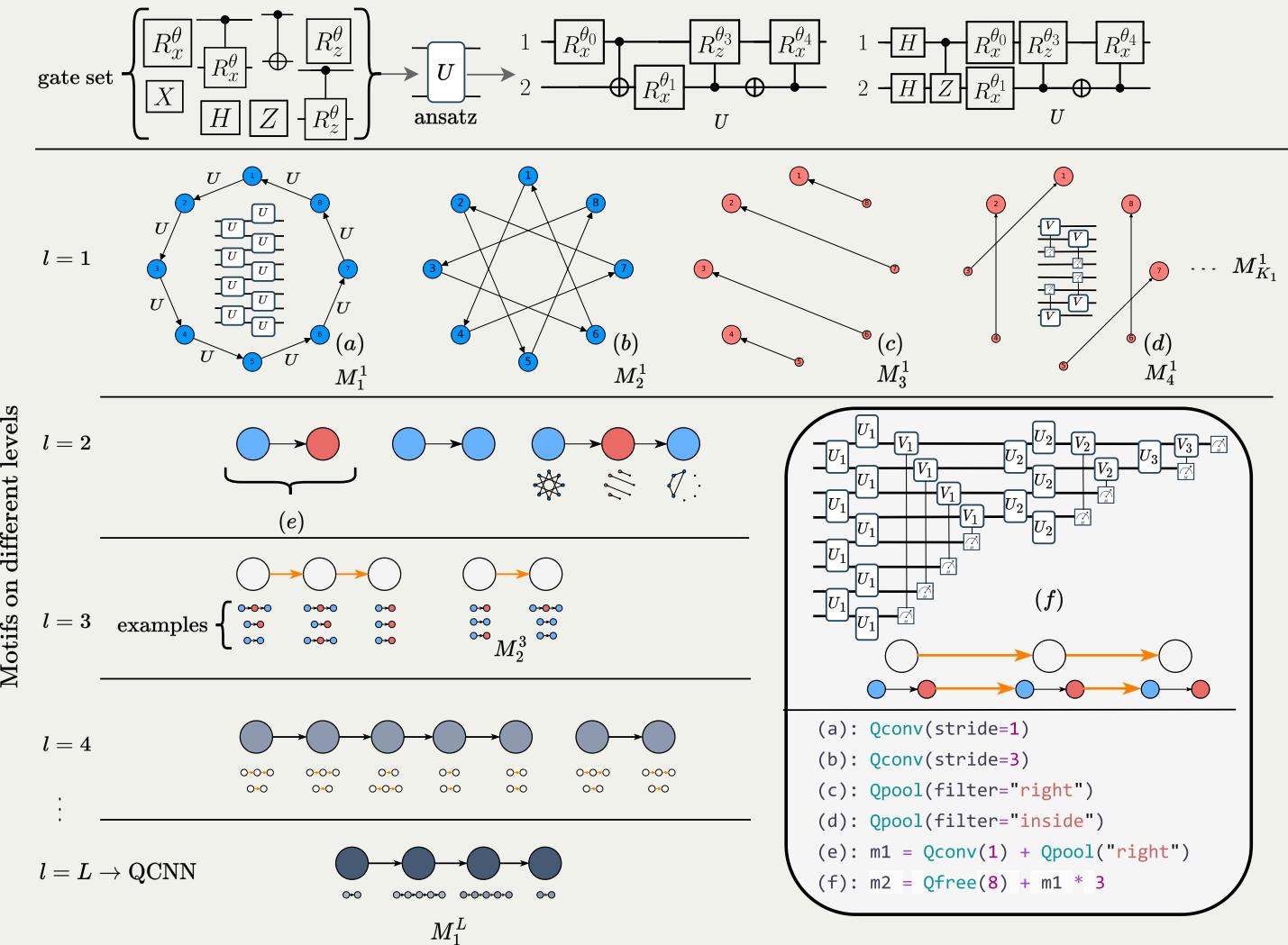
Program Synthesis

- Neural Networks
- Tensor Networks
- Quantum Circuits
- Classical Circuits



Representation

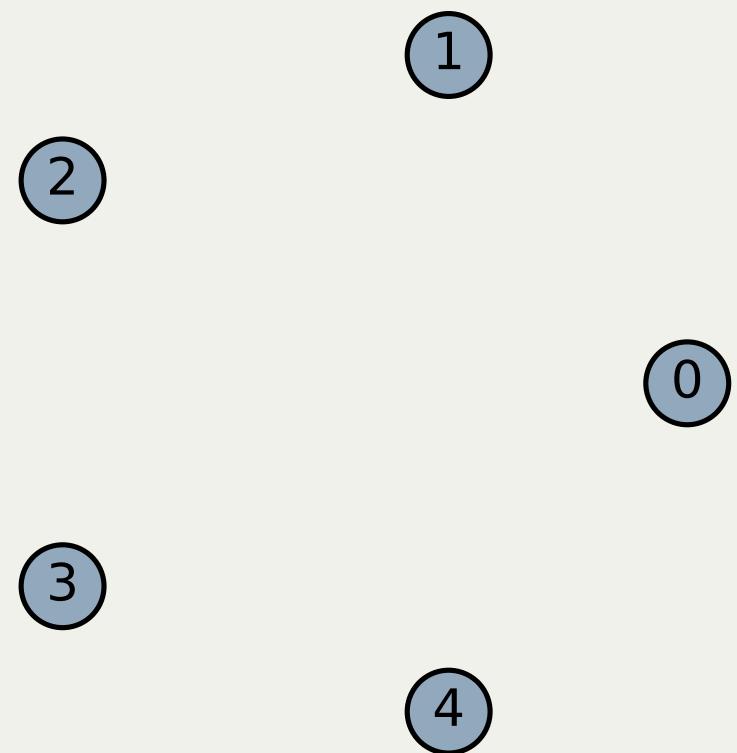
Motifs



¹Lourens, M., Sinayskiy, I., Park, D.K. et al. Hierarchical quantum circuit representations for neural architecture search. npj Quantum Inf 9, 79 (2023).

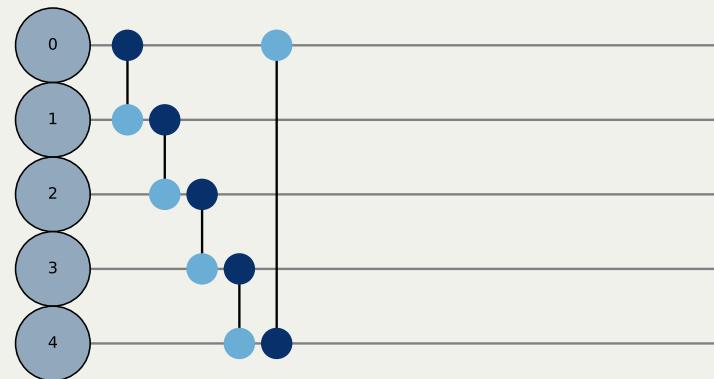
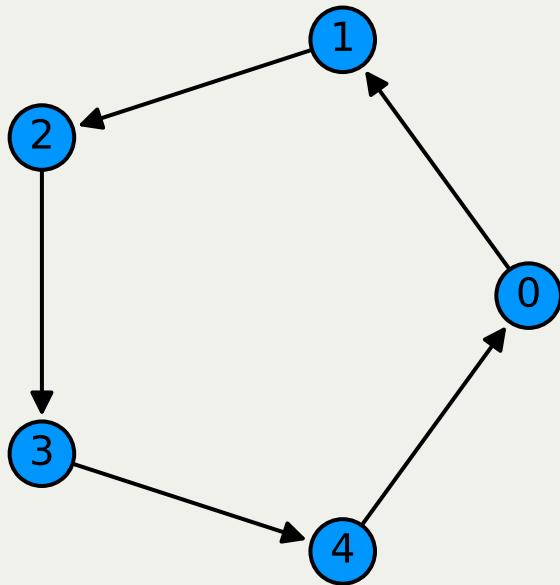
Examples

```
1 hierq = Qinit(5)
2 hierq = Qinit(5) + Qcycle(stride=1, step=1, offset=0)
3 hierq = Qinit(8) + Qcycle(stride=1, step=1, offset=0)
```



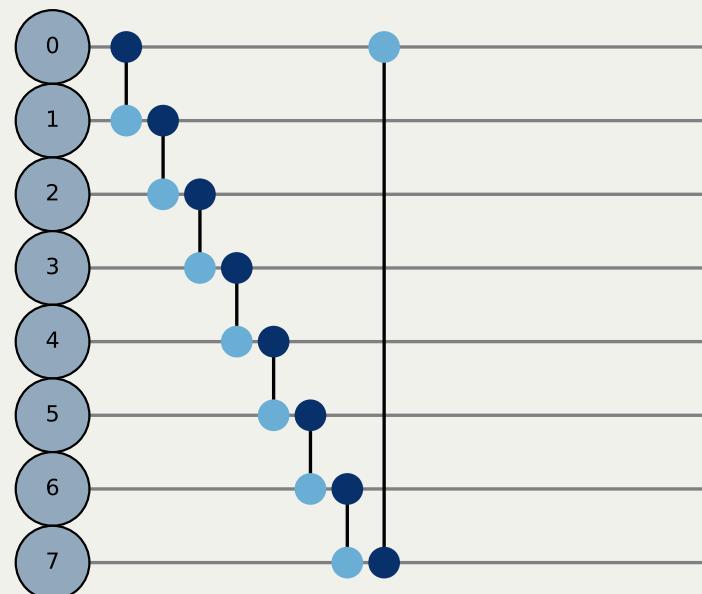
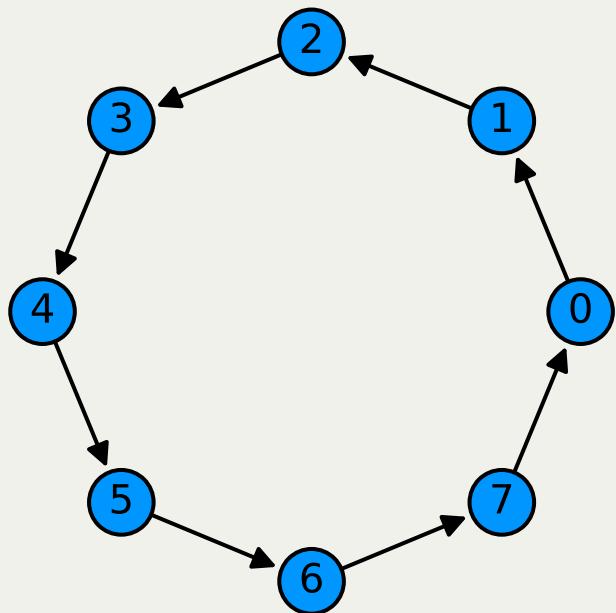
Examples

```
1 hierq = Qinit(5)
2 hierq = Qinit(5) + Qcycle(stride=1, step=1, offset=0)
3 hierq = Qinit(8) + Qcycle(stride=1, step=1, offset=0)
```



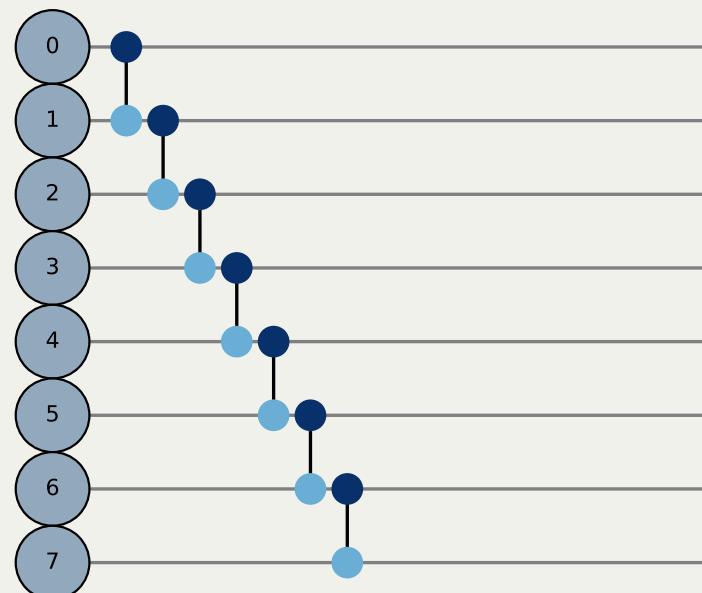
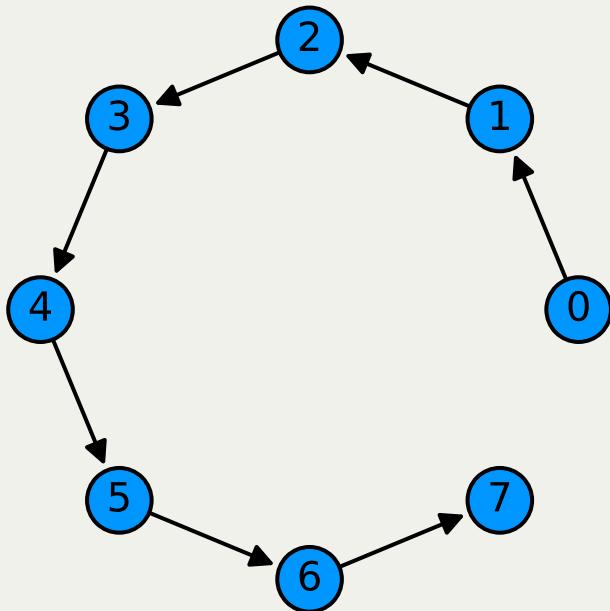
Examples

```
1 hierq = Qinit(5) + Qcycle(stride=1, step=1, offset=0)
2 hierq = Qinit(8) + Qcycle(stride=1, step=1, offset=0)
3 hierq = Qinit(8) + Qcycle(stride=1, step=1, offset=0, boundary="open")
```



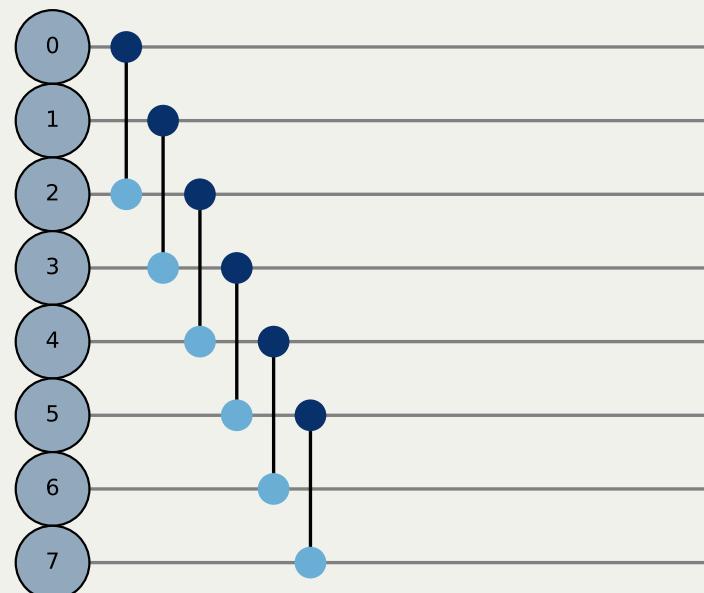
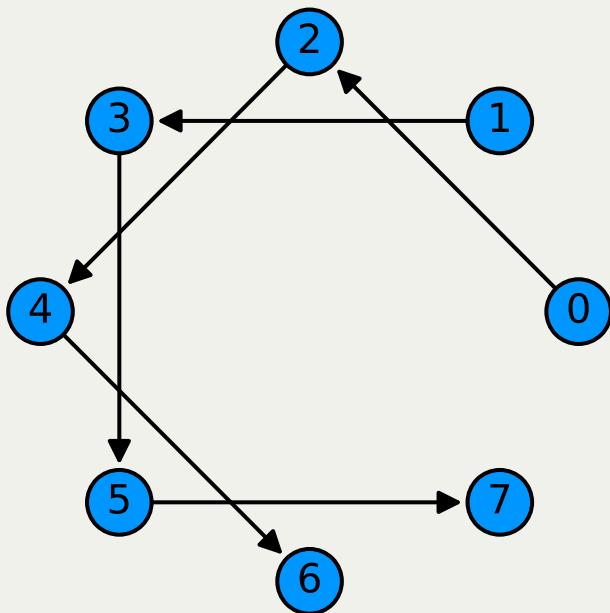
Examples

```
1 hierq = Qinit(8) + Qcycle(stride=1, step=1, offset=0)
2 hierq = Qinit(8) + Qcycle(stride=1, step=1, offset=0, boundary="open")
3 hierq = Qinit(8) + Qcycle(stride=2, step=1, offset=0, boundary="open")
```



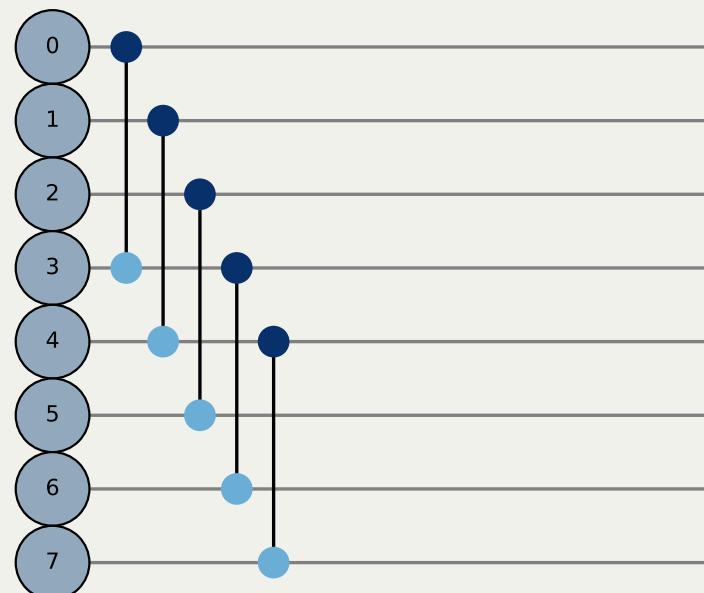
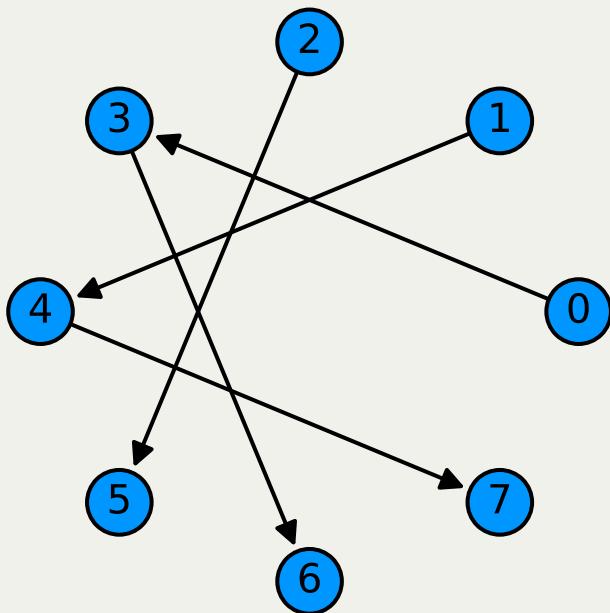
Examples

```
1 hierq = Qinit(8) + Qcycle(stride=1, step=1, offset=0, boundary="open")
2 hierq = Qinit(8) + Qcycle(stride=2, step=1, offset=0, boundary="open")
3 hierq = Qinit(8) + Qcycle(stride=3, step=1, offset=0, boundary="open")
```



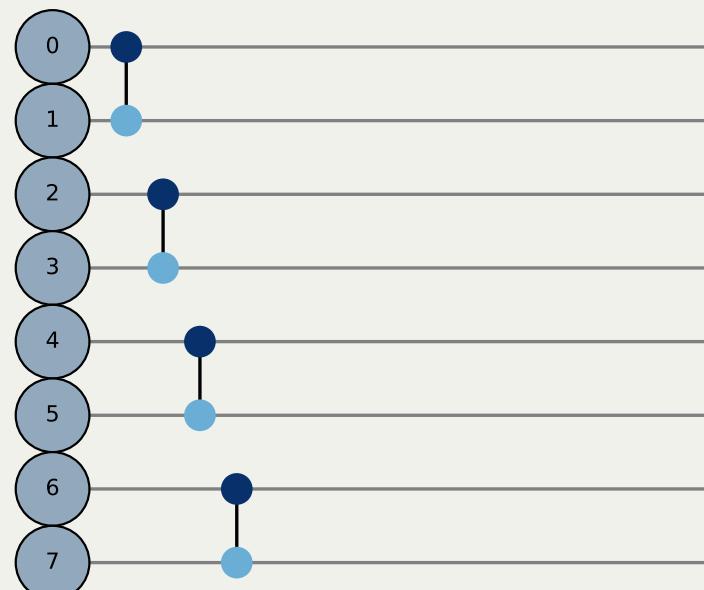
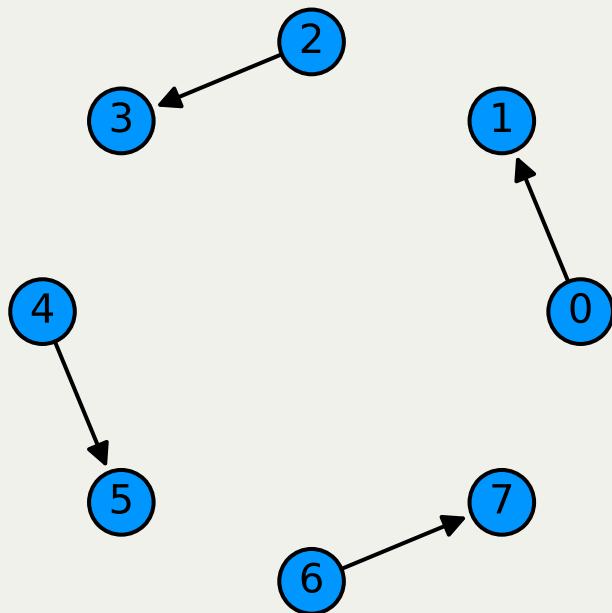
Examples

```
1 hierq = Qinit(8) + Qcycle(stride=2, step=1, offset=0, boundary="open")
2 hierq = Qinit(8) + Qcycle(stride=3, step=1, offset=0, boundary="open")
3 hierq = Qinit(8) + Qcycle(stride=1, step=2, offset=0, boundary="open")
```



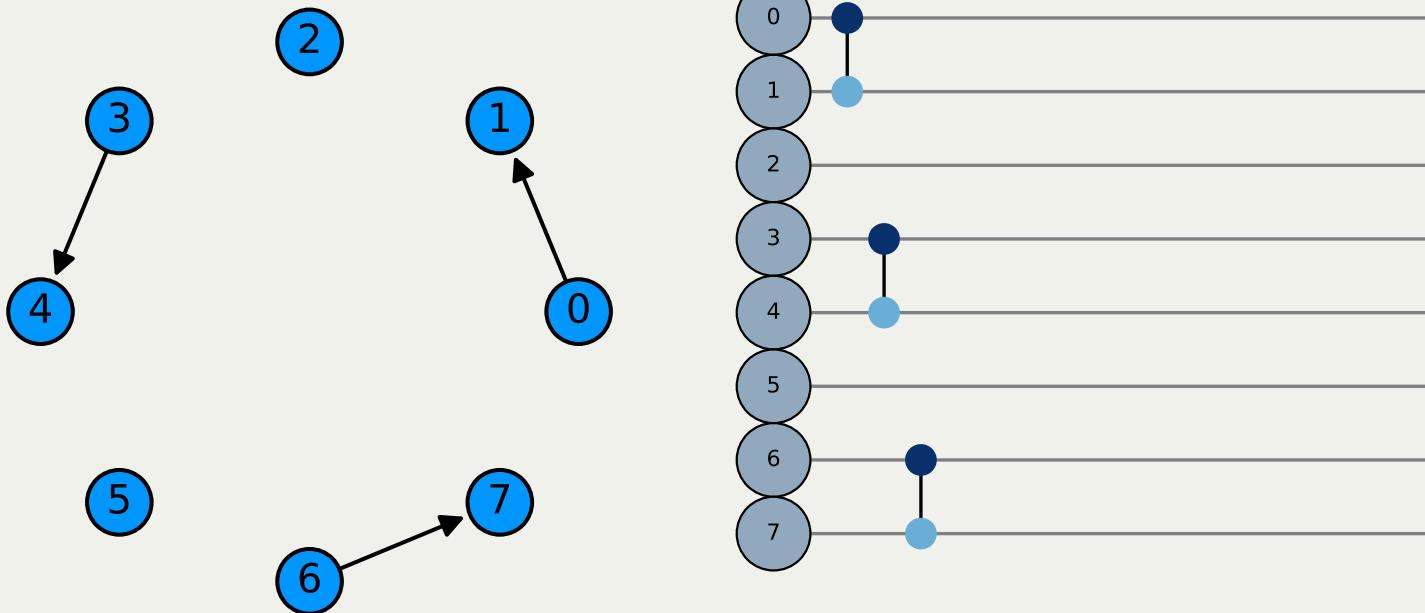
Examples

```
1 hierq = Qinit(8) + Qcycle(stride=3, step=1, offset=0, boundary="open")
2 hierq = Qinit(8) + Qcycle(stride=1, step=2, offset=0, boundary="open")
3 hierq = Qinit(8) + Qcycle(stride=1, step=3, offset=0, boundary="open")
```



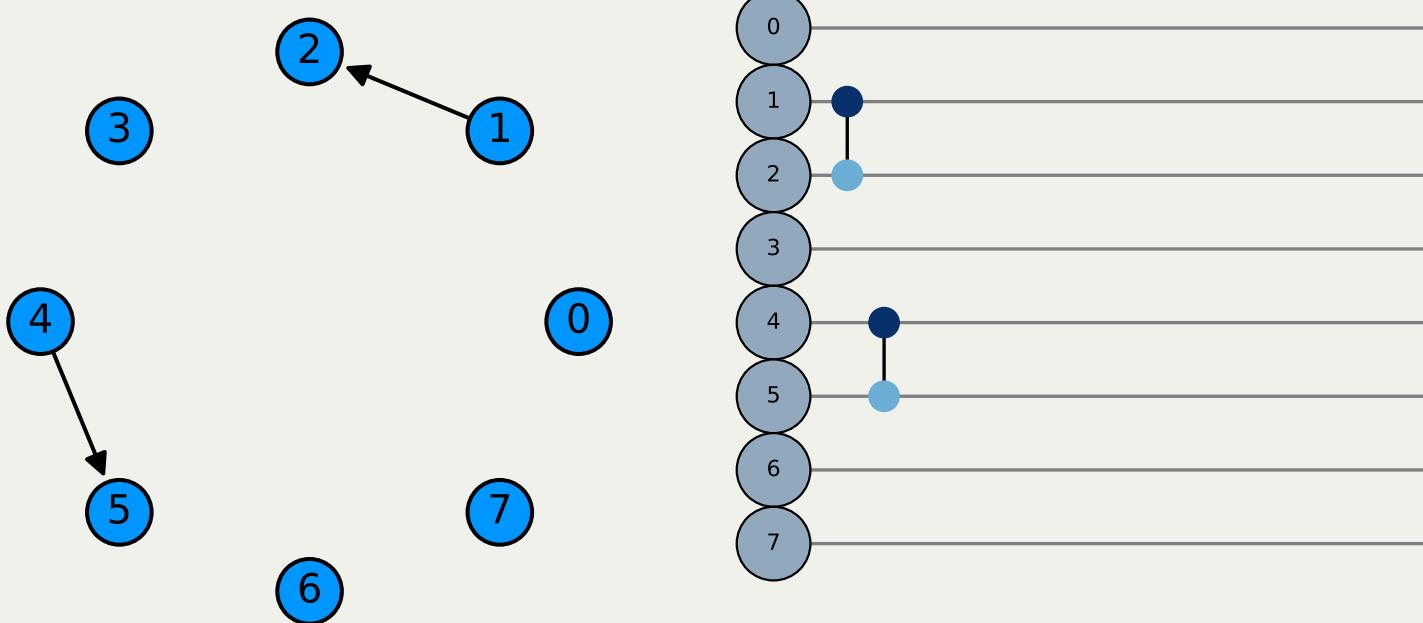
Examples

```
1 hierq = Qinit(8) + Qcycle(stride=1, step=2, offset=0, boundary="open")
2 hierq = Qinit(8) + Qcycle(stride=1, step=3, offset=0, boundary="open")
3 hierq = Qinit(8) + Qcycle(stride=1, step=3, offset=1, boundary="open")
```



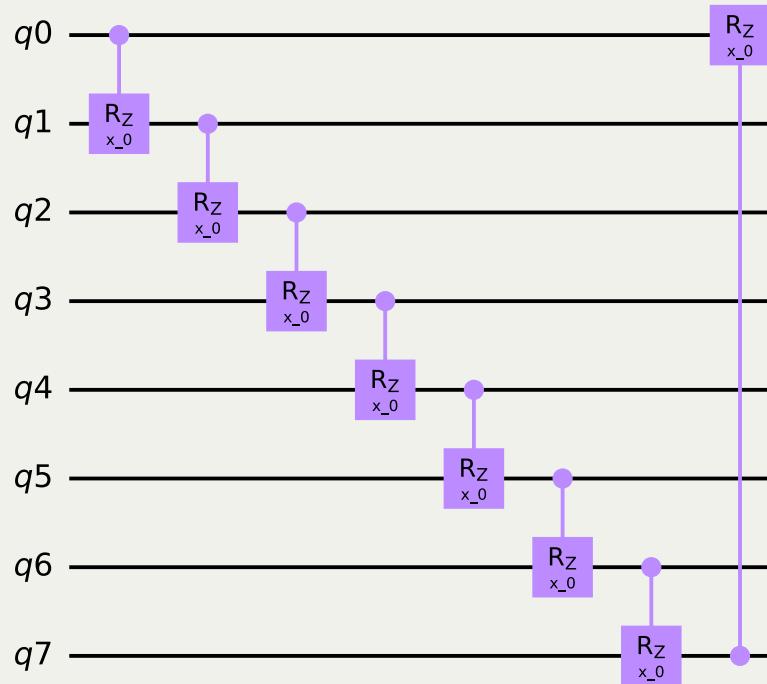
Examples

```
1 hierq = Qinit(8) + Qcycle(stride=1, step=3, offset=0, boundary="open")
2 hierq = Qinit(8) + Qcycle(stride=1, step=3, offset=1, boundary="open")
3 hierq = Qinit(8) + Qcycle(1,1,0, mapping = u2)
```



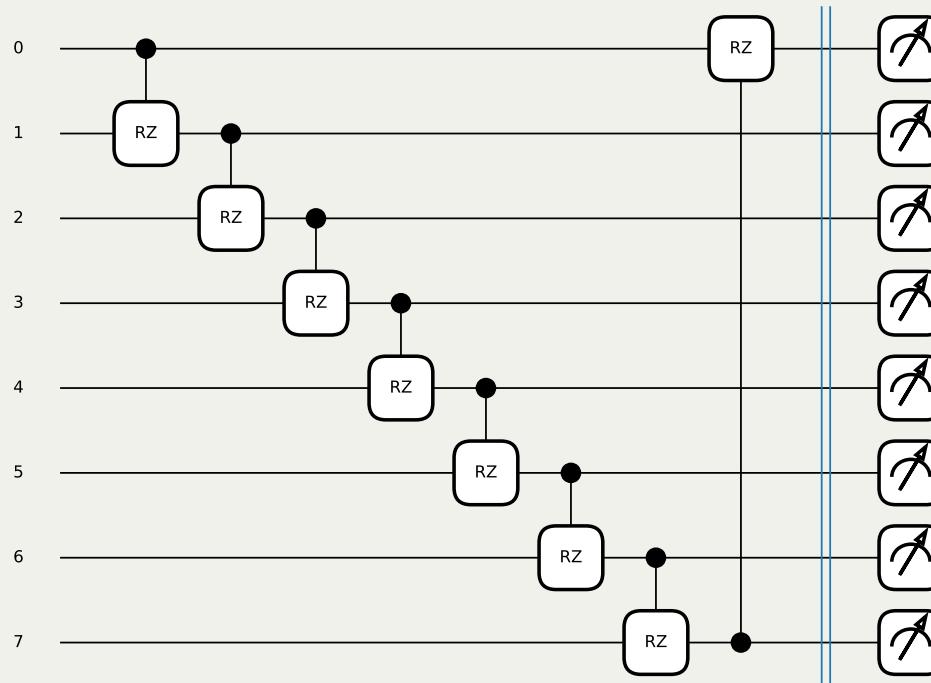
Examples

```
1 hierq = Qinit(8) + Qcycle(1,1,0, mapping = u2)
2 hierq(backend="qiskit")
3 hierq(backend="pennylane")
```



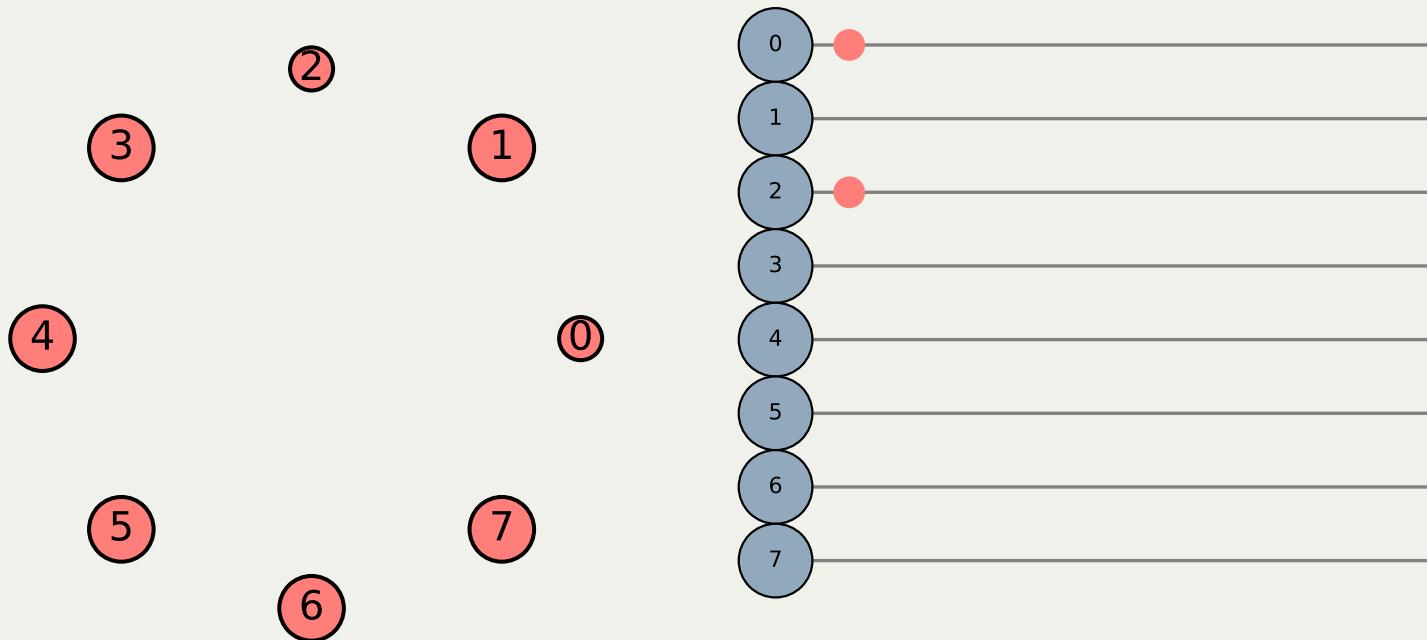
Examples

```
1 hierq = Qinit(8) + Qcycle(1,1,0, mapping = u2)
2 hierq(backend="qiskit")
3 hierq(backend="pennylane")
```



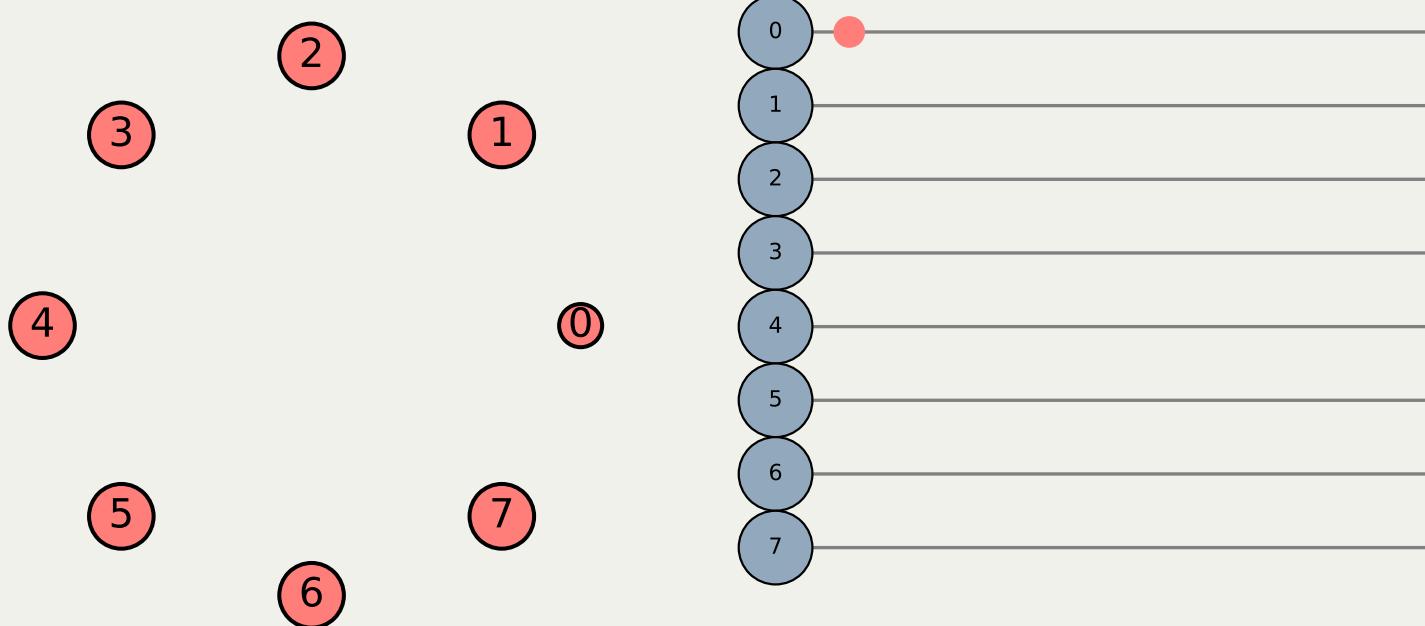
Examples

```
1 hierq = Qinit(8) + Qmask("10100000")
2 hierq = Qinit(8) + Qmask("1*")
3 hierq = Qinit(8) + Qmask("!00")
```



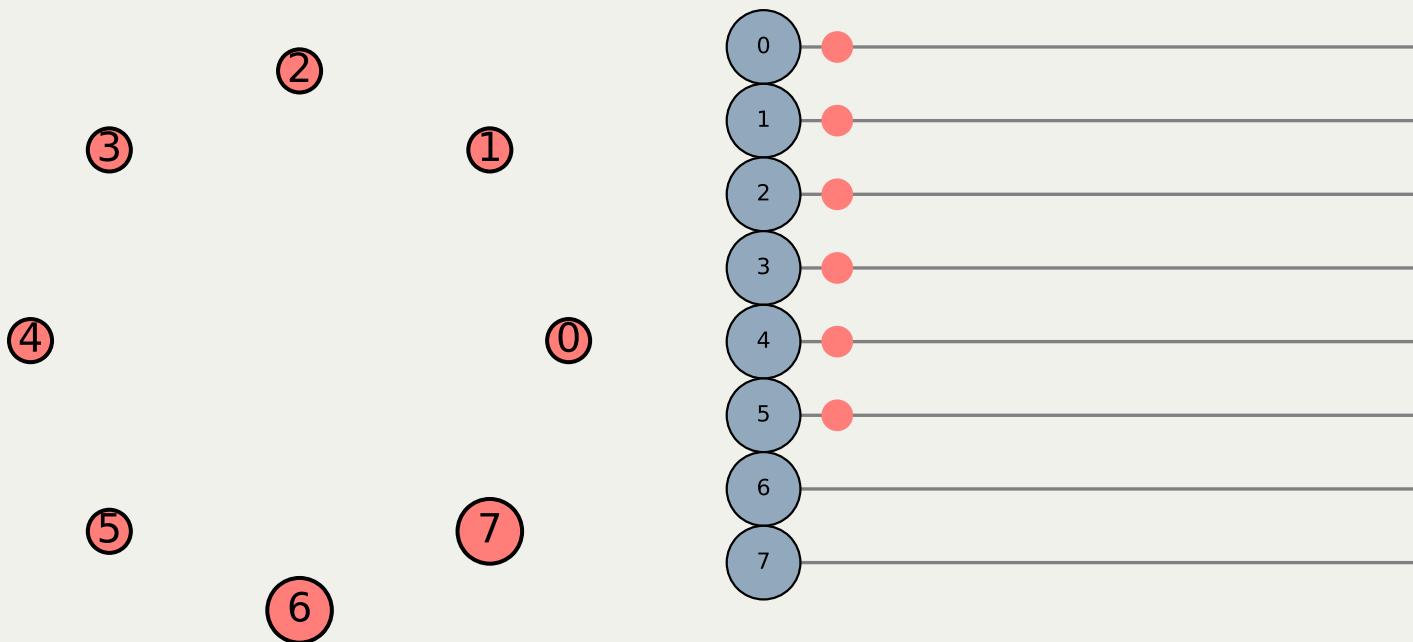
Examples

```
1 hierq = Qinit(8) + Qmask("10100000")
2 hierq = Qinit(8) + Qmask("1*")
3 hierq = Qinit(8) + Qmask("!00")
```



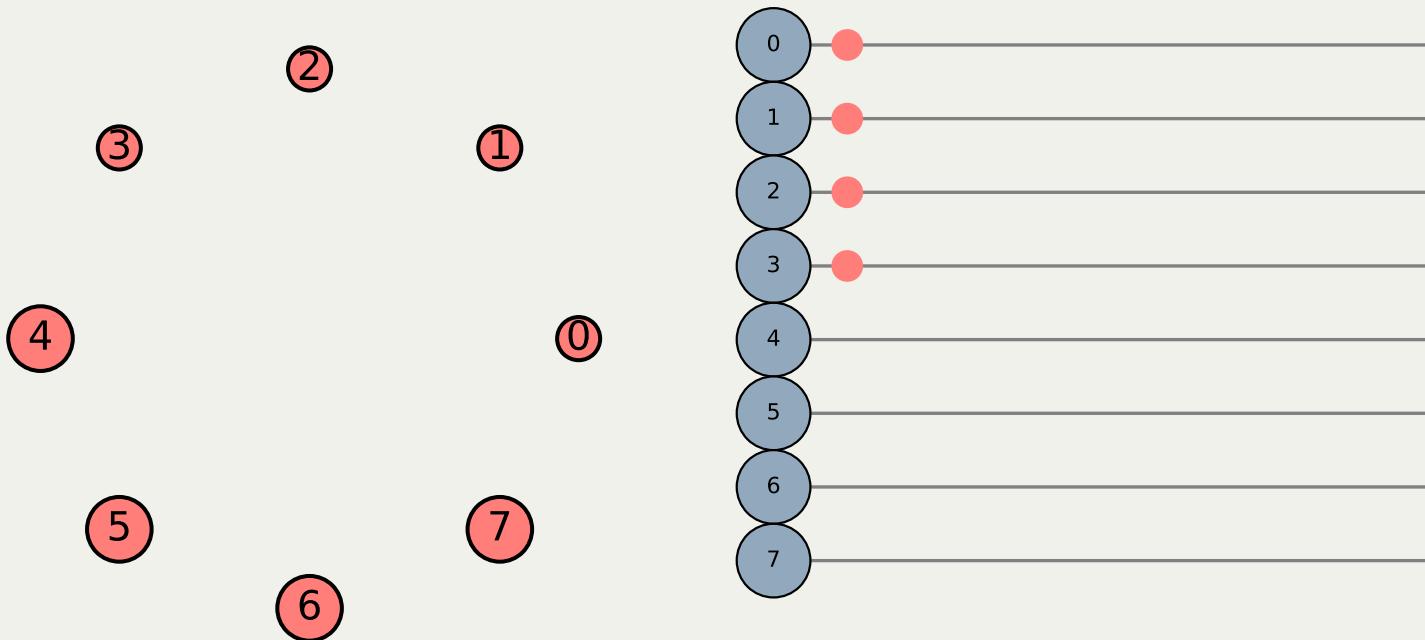
Examples

```
1 hierq = Qinit(8) + Qmask("1*")
2 hierq = Qinit(8) + Qmask("!00")
3 hierq = Qinit(8) + Qmask("!*")
```



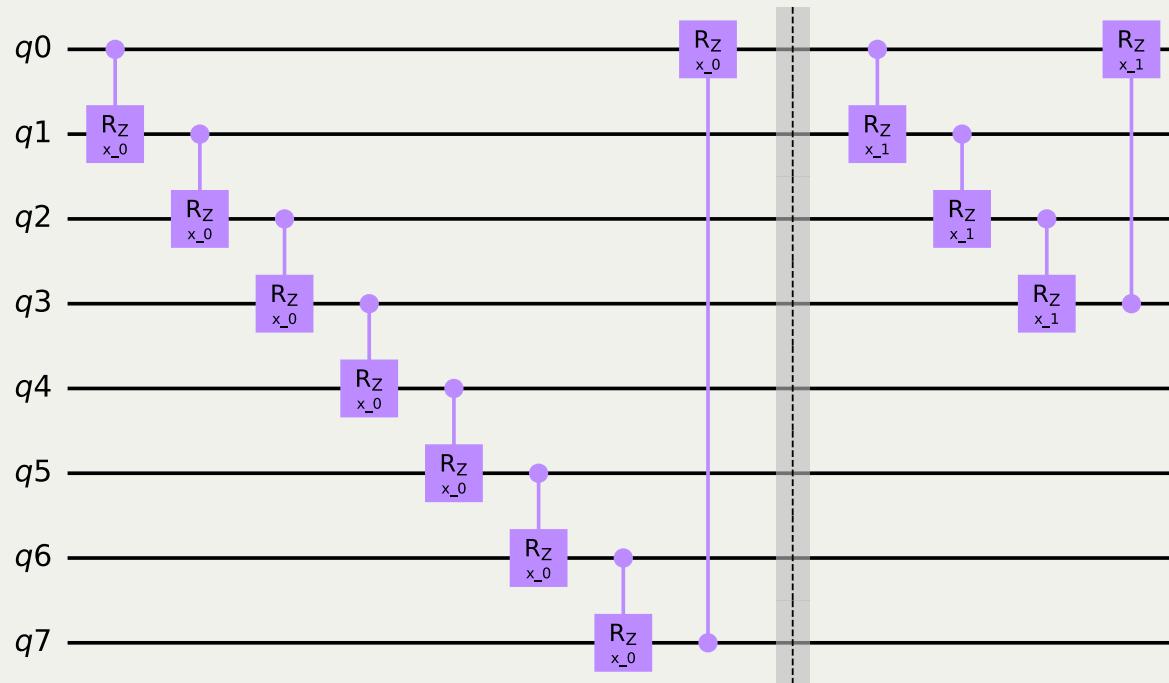
Examples

```
1 hierq = Qinit(8) + Qmask("1*")
2 hierq = Qinit(8) + Qmask("!00")
3 hierq = Qinit(8) + Qcycle(mapping = u2) + Qmask("*!") + Qcycle(mapping = u2)
```



Examples

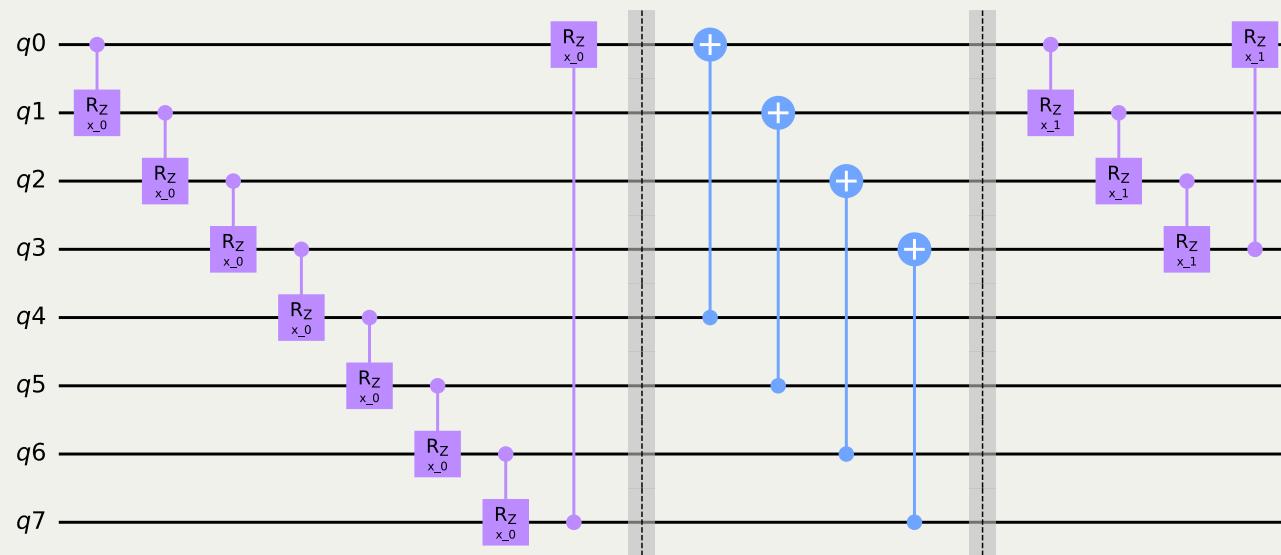
```
1 hierq = Qinit(8) + Qcycle(mapping = u2) + Qmask("*!") + Qcycle(mapping = u2)
```



<https://github.com/matt-lourens/hierarqcal>

Examples

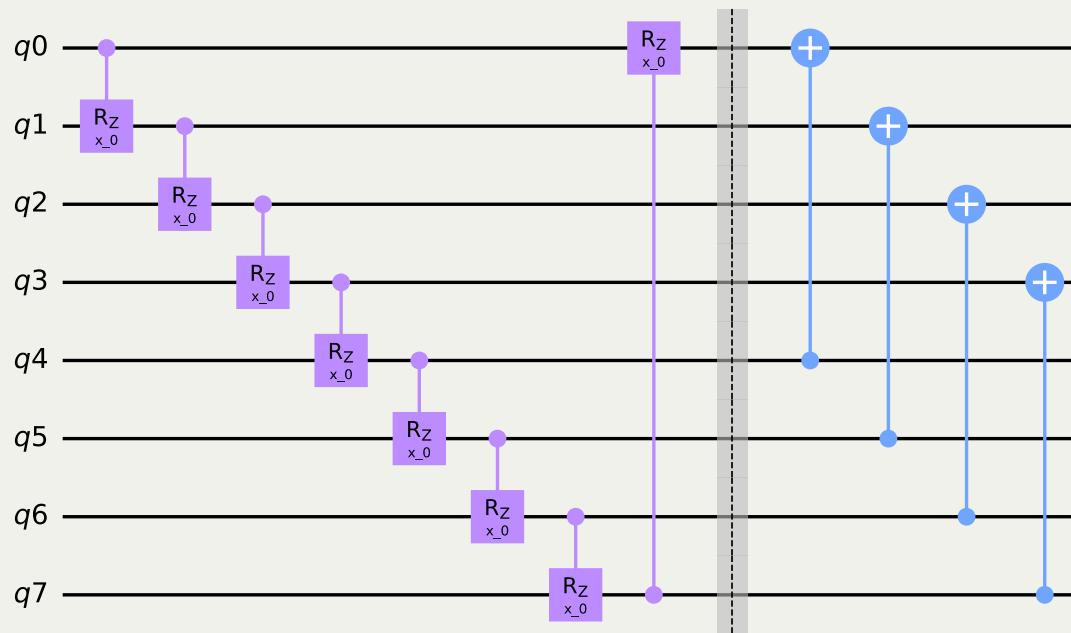
```
1 cycle = Qcycle(mapping = u2)
2 hierq = Qinit(8) + cycle + Qmask("*!", mapping=v2) + cycle
```



<https://github.com/matt-lourens/hierarqcal>

Examples

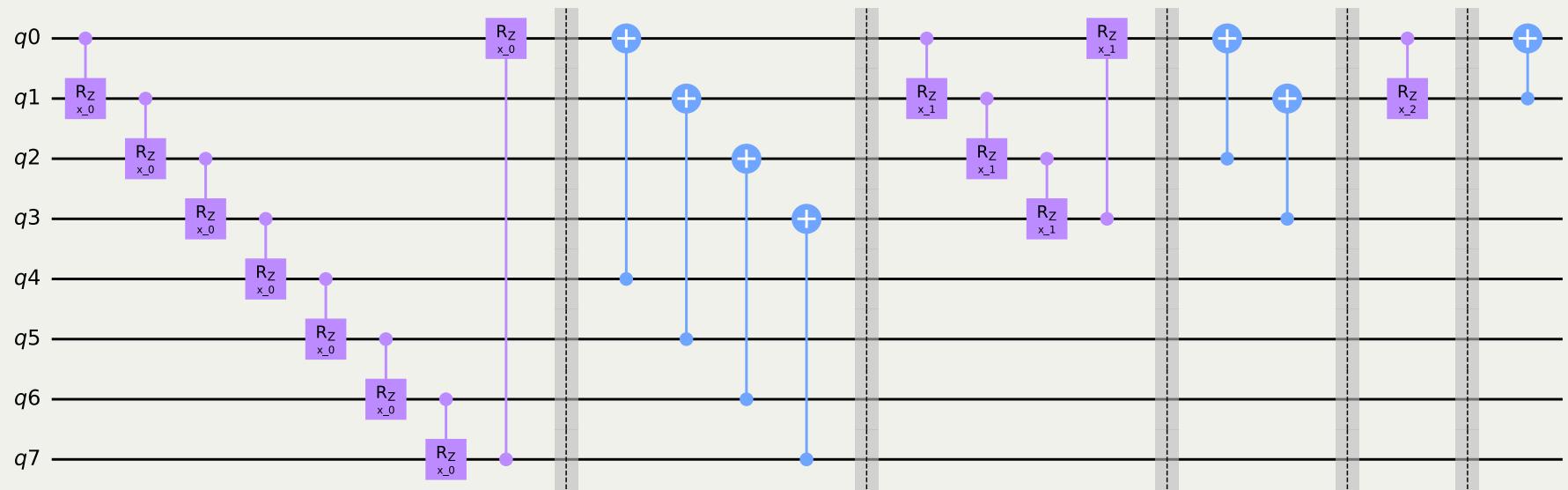
```
1 cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("*!", mapping=v2)
2 hierq = Qinit(8) + cycle_mask
3 hierq = Qinit(8) + cycle_mask * 3
```



<https://github.com/matt-lourens/hierarqcal>

Examples

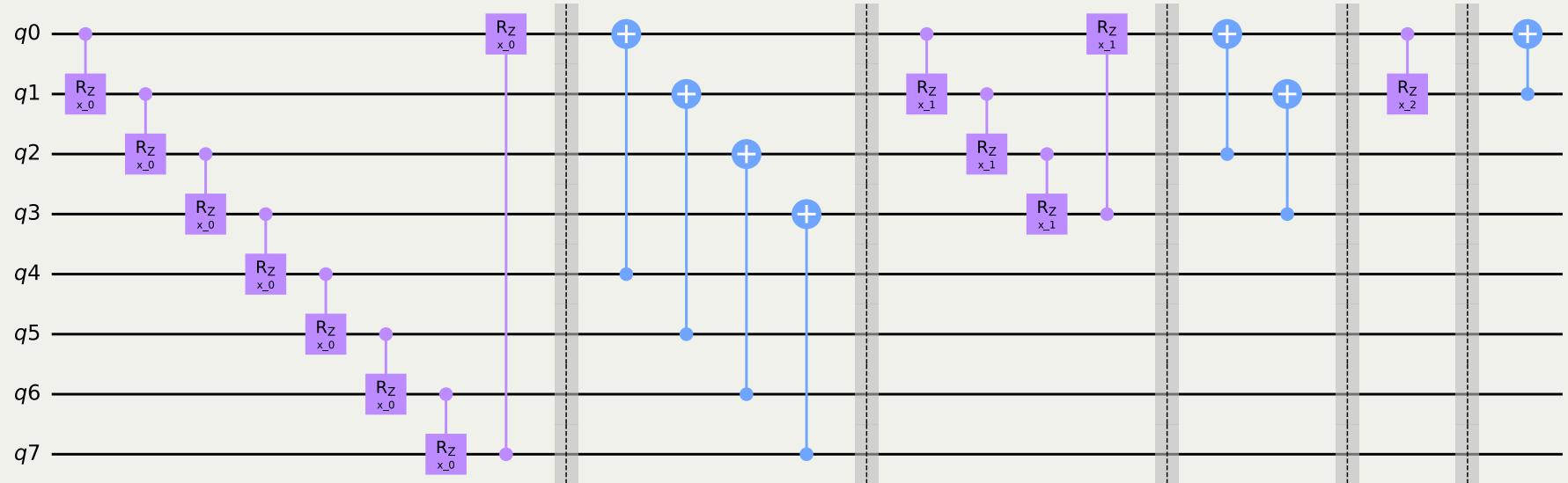
```
1 cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("*!", mapping=v2)
2 hierq = Qinit(8) + cycle_mask
3 hierq = Qinit(8) + cycle_mask * 3
```



<https://github.com/matt-lourens/hierarqcal>

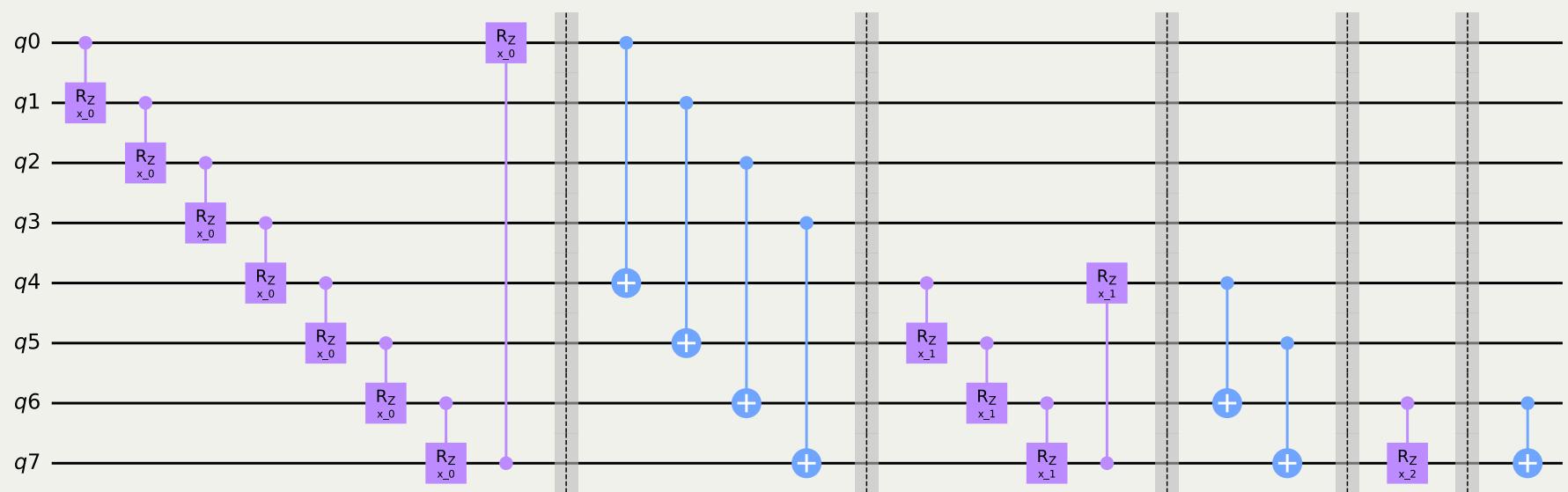
Examples

```
1 cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("*!", mapping=v2)
2 cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("!*", mapping=v2)
3 cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("!*!", mapping=v2)
4 cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("*!*", mapping=v2)
5 hierq = Qinit(8) + cycle_mask * 3
```



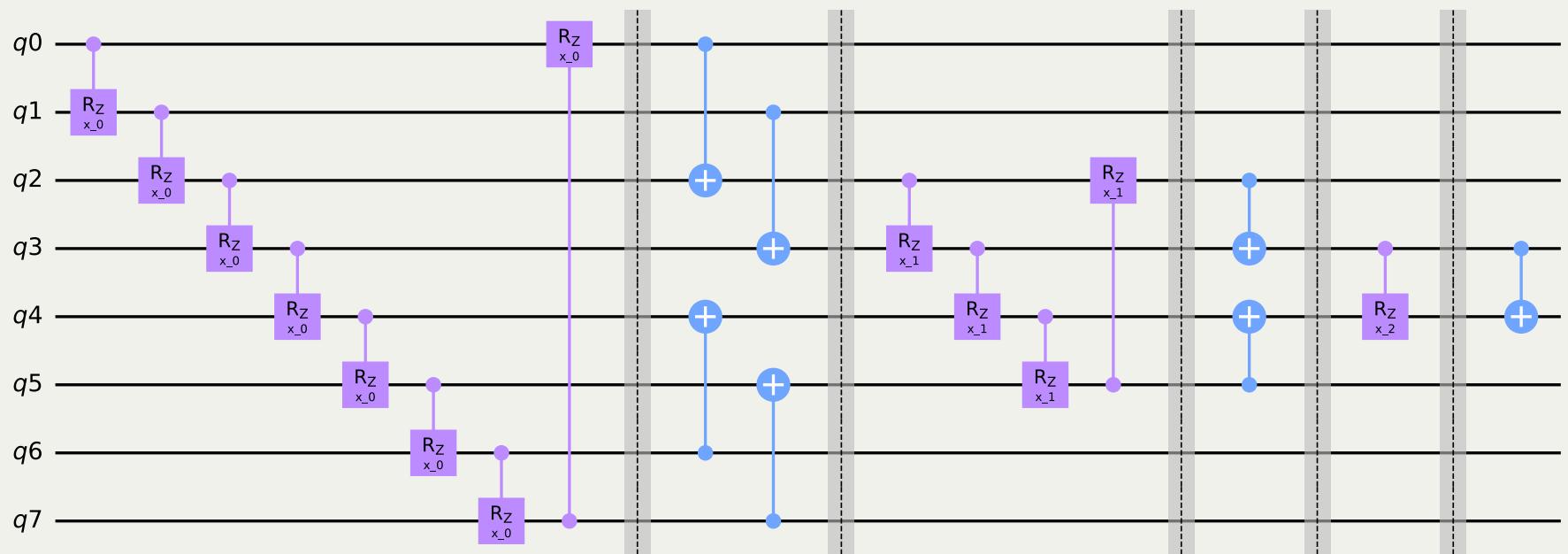
Examples

```
1 cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("*!", mapping=v2)
2 cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("!*", mapping=v2)
3 cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("!*!", mapping=v2)
4 cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("*!*", mapping=v2)
5 hierq = Qinit(8) + cycle_mask * 3
```



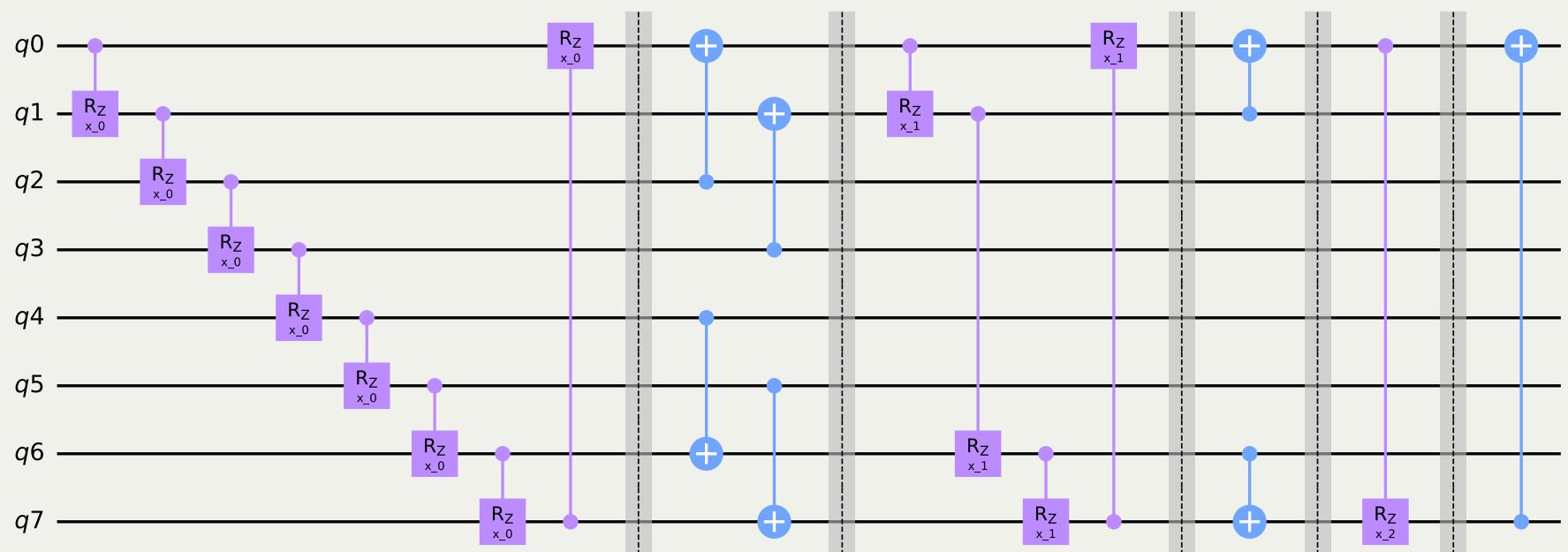
Examples

```
1 cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("*!*", mapping=v2)
2 cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("!*!", mapping=v2)
3 cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("!*!", mapping=v2)
4 cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("*!*", mapping=v2)
5 hierq = Qinit(8) + cycle_mask * 3
```



Examples

```
1 cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("*!", mapping=v2)
2 cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("!*", mapping=v2)
3 cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("!*!", mapping=v2)
4 cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("*!*", mapping=v2)
5 hierq = Qinit(8) + cycle_mask * 3
```



Results and Usage

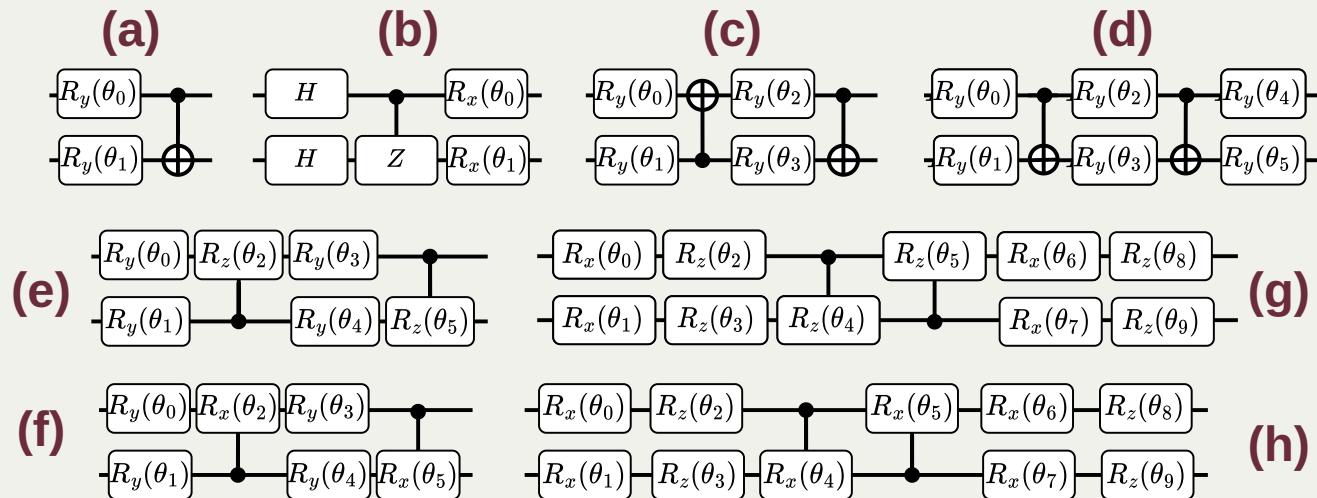
Random Search

Random Search

```

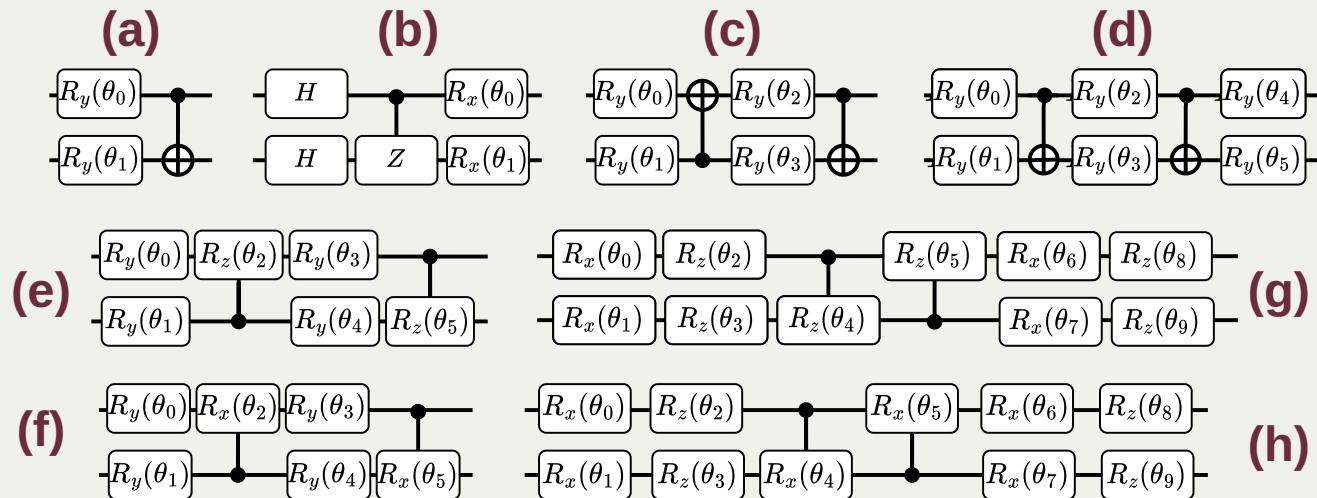
1 # Example Search Space
2 strides = range(1, 8, 1)
3 steps = range(1, 8, 1)
4 mappings = [a, b, c, d, e, f, g, h]
5 boundaries = ["open", "periodic"]
6 patterns = ["10", "01", "*!", "!*", "!*!", "*!*"]
7
8 qcnn = Qinit(8) + (Qcycle(...) + Qmask(...)) * 3

```



Random Search

```
1 # Example Search Space
2 strides = range(1, 8, 1)
3 steps = range(1, 8, 1)
4 mappings = [a, b, c, d, e, f, g, h]
5 boundaries = ["open", "periodic"]
6 patterns = ["10", "01", "*!", "!*", "!*!", "*!*"]
7
8 qcnn = Qinit(8) + (Qcycle(...) + Qmask(...)) * 3
```

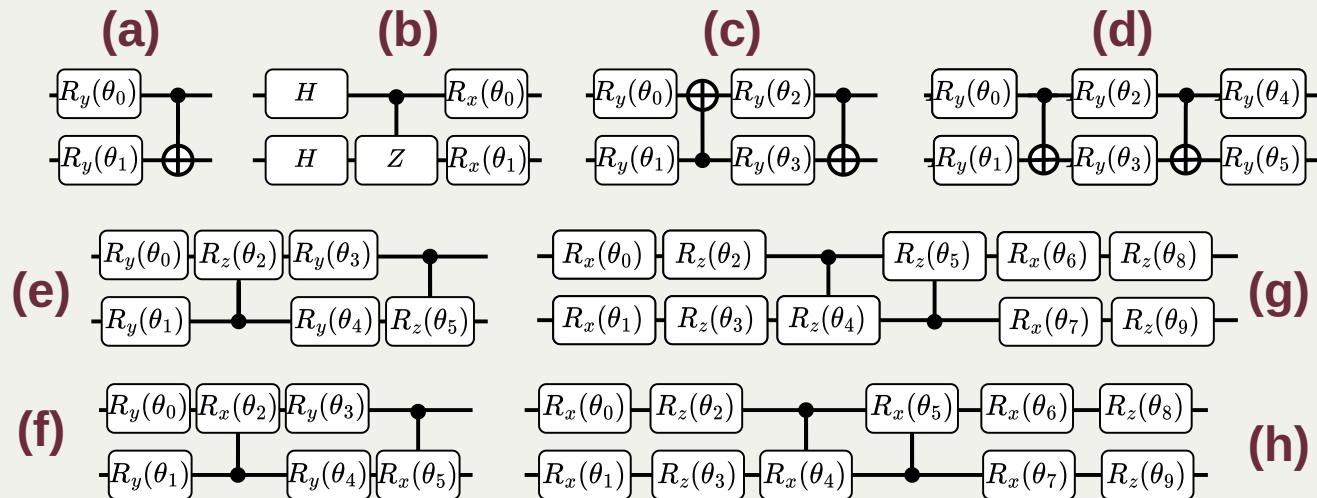


Random Search

```

1 # Example Search Space
2 strides = range(1, 8, 1)
3 steps = range(1, 8, 1)
4 mappings = [a, b, c, d, e, f, g, h]
5 boundaries = ["open", "periodic"]
6 patterns = ["10", "01", "*!", "!*", "!*!", "*!*"]
7
8 qcnn = Qinit(8) + (Qcycle(...) + Qmask(...)) * 3

```

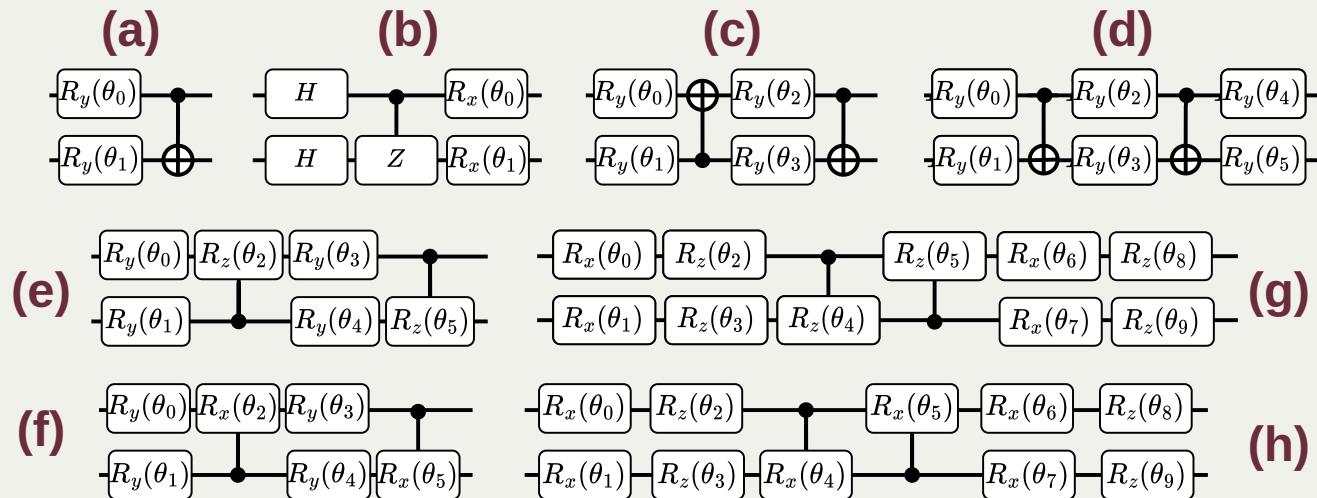


Random Search

```

1 # Example Search Space
2 strides = range(1, 8, 1)
3 steps = range(1, 8, 1)
4 mappings = [a, b, c, d, e, f, g, h]
5 boundaries = ["open", "periodic"]
6 patterns = ["10", "01", "*!", "!*", "!*!", "*!*"]
7
8 qcnn = Qinit(8) + (Qcycle(...) + Qmask(...)) * 3

```

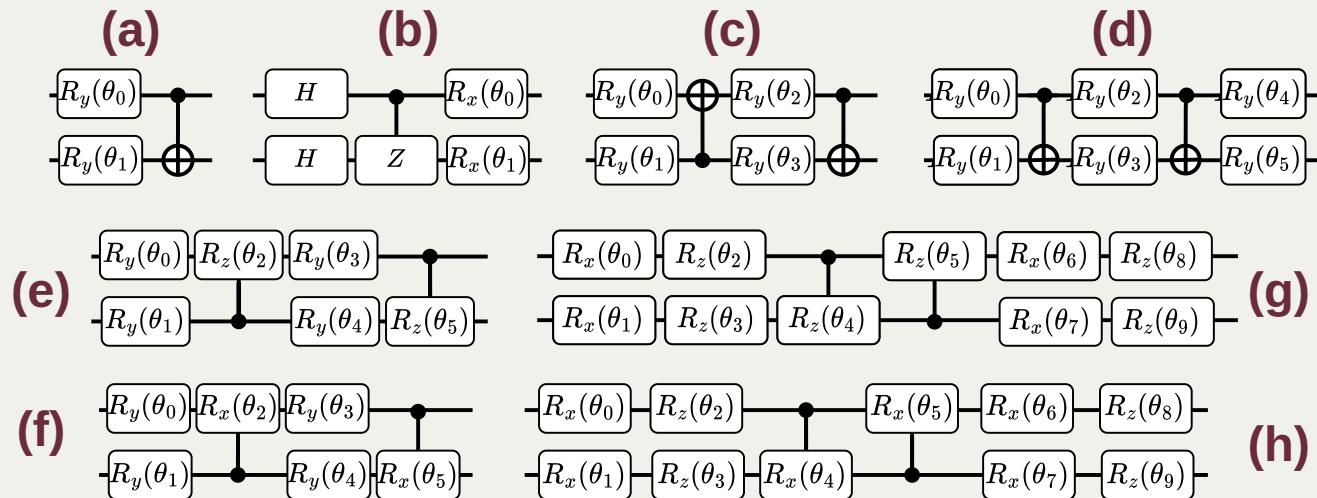


Random Search

```

1 # Example Search Space
2 strides = range(1, 8, 1)
3 steps = range(1, 8, 1)
4 mappings = [a, b, c, d, e, f, g, h]
5 boundaries = ["open", "periodic"]
6 patterns = ["10", "01", "*!", "!*", "!*!", "*!*"]
7
8 qcnn = Qinit(8) + (Qcycle(...) + Qmask(...)) * 3

```

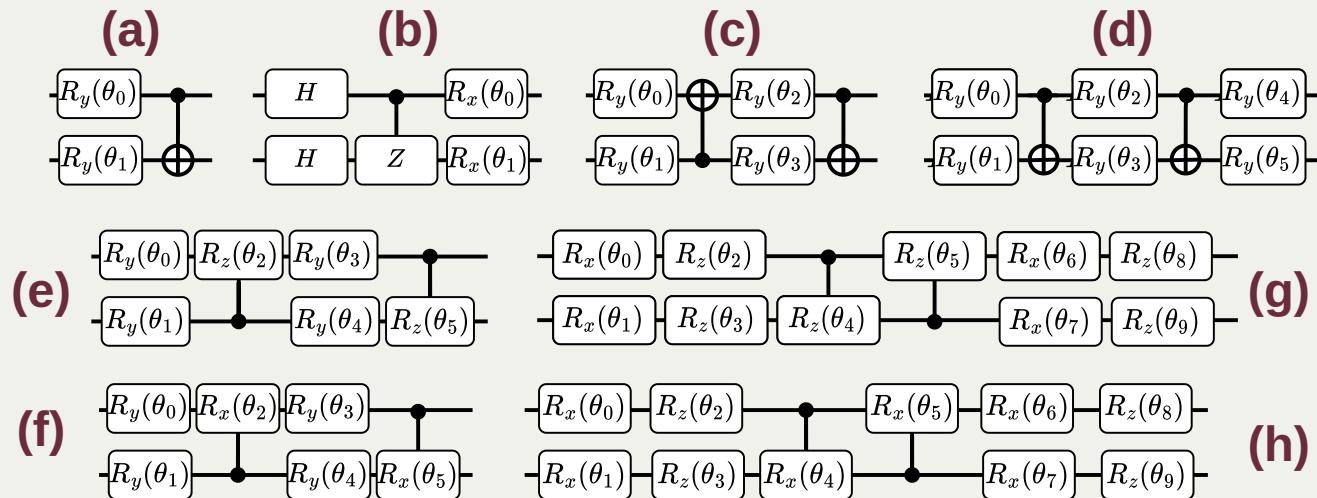


Random Search

```

1 # Example Search Space
2 strides = range(1, 8, 1)
3 steps = range(1, 8, 1)
4 mappings = [a, b, c, d, e, f, g, h]
5 boundaries = ["open", "periodic"]
6 patterns = ["10", "01", "*!", "!*", "!*!", "*!*"]
7
8 qcnn = Qinit(8) + (Qcycle(...) + Qmask(...)) * 3

```

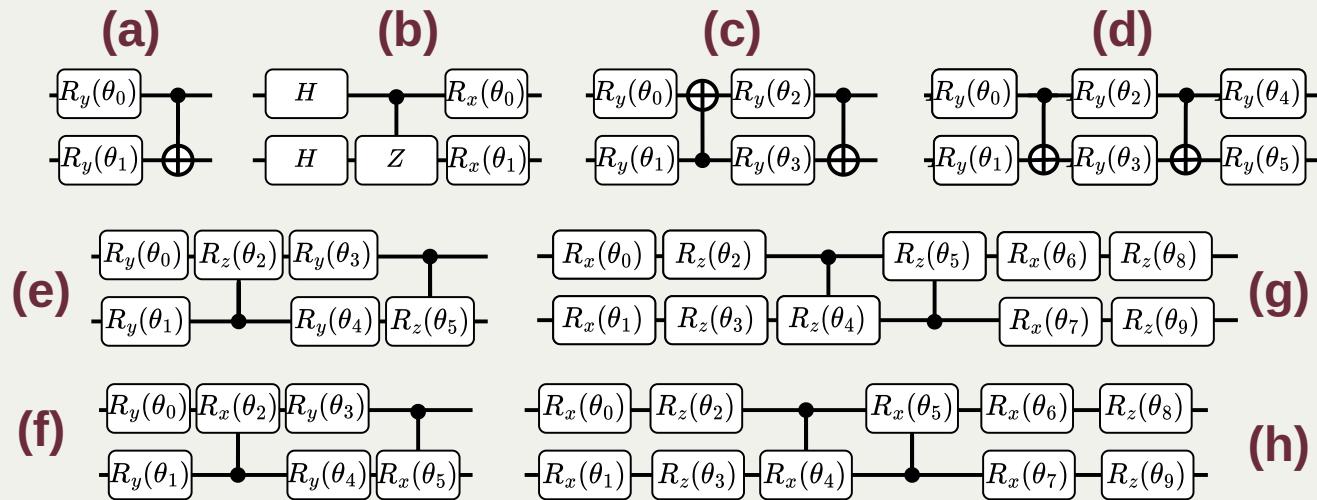


Random Search

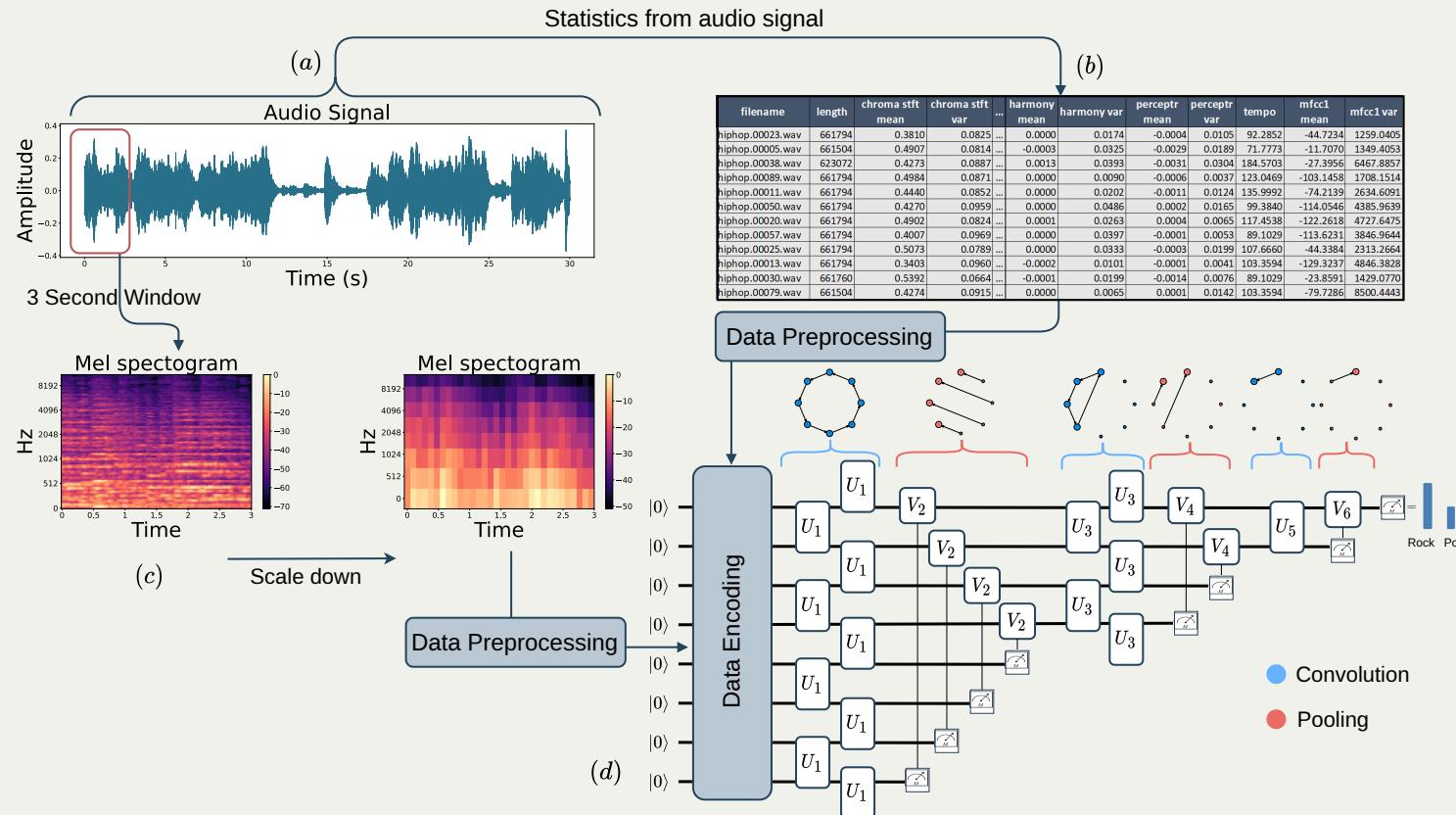
```

1 # Example Search Space
2 strides = range(1, 8, 1)
3 steps = range(1, 8, 1)
4 mappings = [a, b, c, d, e, f, g, h]
5 boundaries = ["open", "periodic"]
6 patterns = ["10", "01", "*!", "!*", "!*!", "*!*"]
7
8 qcnn = Qinit(8) + (Qcycle(...) + Qmask(...)) * 3

```

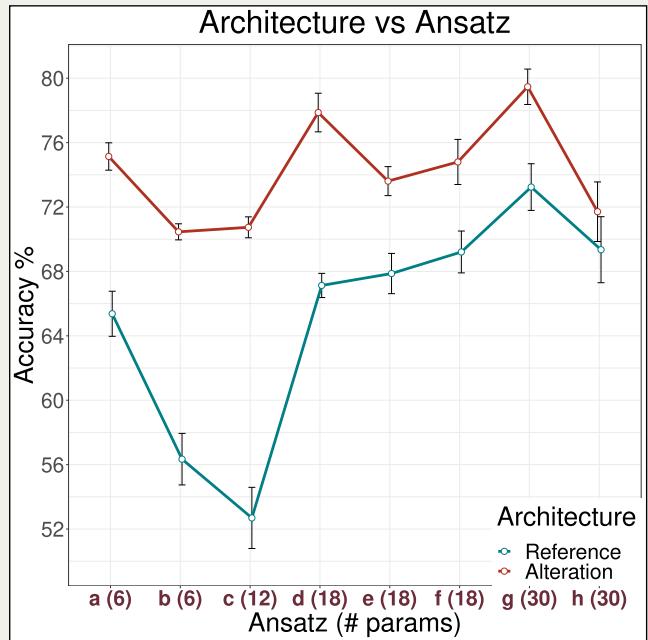


Music Genre Classification



Lourens, M., Sinayskiy, I., Park, D.K. et al. Hierarchical quantum circuit representations for neural architecture search. npj Quantum Inf 9, 79 (2023).

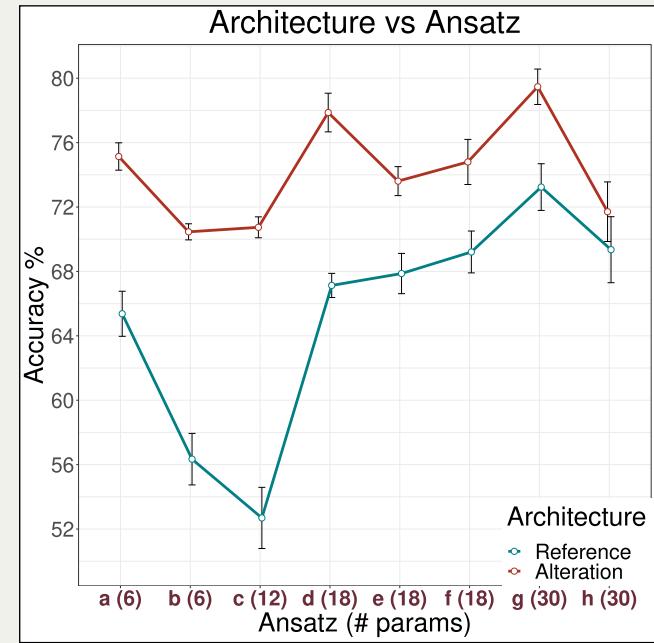
Music Genre Classification



Lourens, M., Sinayskiy, I., Park, D.K. et al. Hierarchical quantum circuit representations for neural architecture search. *npj Quantum Inf* 9, 79 (2023).

*Hur, T., Kim, L. & Park, D. K. Quantum convolutional neural network for classical data classification. *Quantum Mach. Intell.* 4, 3 (2022).

Music Genre Classification

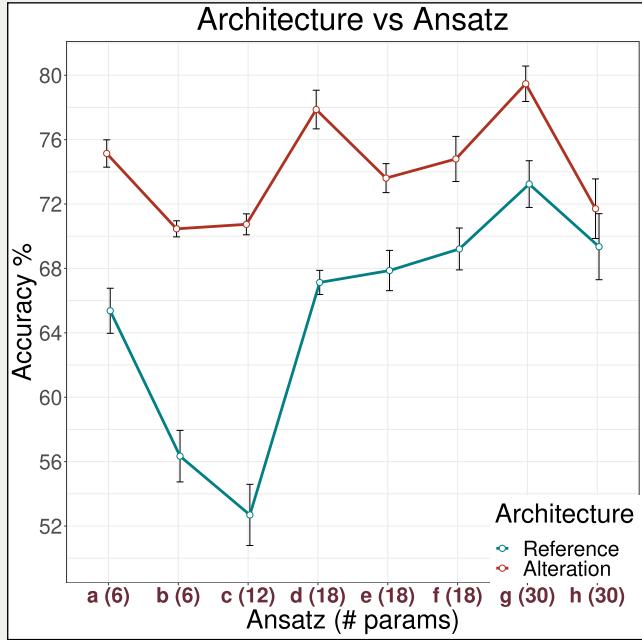


Architecture vs Ansatz					
Ansatz, # Params	Architecture		Δ	Alteration (s_c, F^*, s_p)	
	Reference	New alteration		(s_c, F^*, s_p)	
a, 6	65.37 ± 2.8	75.14 ± 1.7	+9.77	(6, left, 2)	
b, 6	56.34 ± 3.2	70.46 ± 1.0	+14.12	(1, odd, 3)	
c, 12	52.69 ± 3.8	70.74 ± 1.3	+18.05	(1, odd, 0)	
d, 18	67.13 ± 1.5	77.87 ± 2.4	+9.87	(1, outside, 2)	
e, 18	67.87 ± 2.5	73.61 ± 1.8	+5.74	(6, left, 0)	
f, 18	69.21 ± 2.6	74.80 ± 2.8	+5.59	(1, left, 3)	
g, 30	73.24 ± 2.9	79.47 ± 2.2	+6.23	(2, left, 1)	
h, 30	69.35 ± 4.1	71.71 ± 3.7	+2.36	(2, left, 1)	

Lourens, M., Sinayskiy, I., Park, D.K. et al. Hierarchical quantum circuit representations for neural architecture search. npj Quantum Inf 9, 79 (2023).

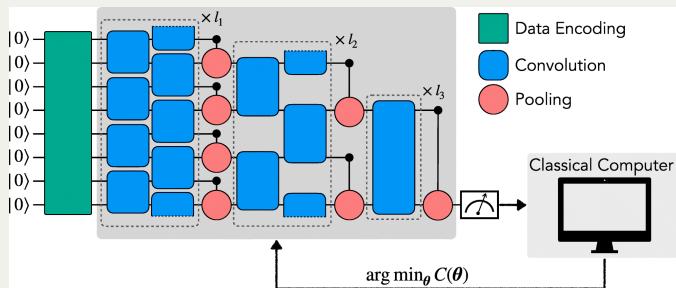
*Hur, T., Kim, L. & Park, D. K. Quantum convolutional neural network for classical data classification. Quantum Mach. Intell. 4, 3 (2022).

Music Genre Classification



Architecture vs Ansatz				
Ansatz, # Params	Architecture		Δ	Alteration (s_c, F^*, s_p)
	Reference	New alteration		
a, 6	65.37 ± 2.8	75.14 ± 1.7	+9.77	(6, left, 2)
b, 6	56.34 ± 3.2	70.46 ± 1.0	+14.12	(1, odd, 3)
c, 12	52.69 ± 3.8	70.74 ± 1.3	+18.05	(1, odd, 0)
d, 18	67.13 ± 1.5	77.87 ± 2.4	+9.87	(1, outside, 2)
e, 18	67.87 ± 2.5	73.61 ± 1.8	+5.74	(6, left, 0)
f, 18	69.21 ± 2.6	74.80 ± 2.8	+5.59	(1, left, 3)
g, 30	73.24 ± 2.9	79.47 ± 2.2	+6.23	(2, left, 1)
h, 30	69.35 ± 4.1	71.71 ± 3.7	+2.36	(2, left, 1)

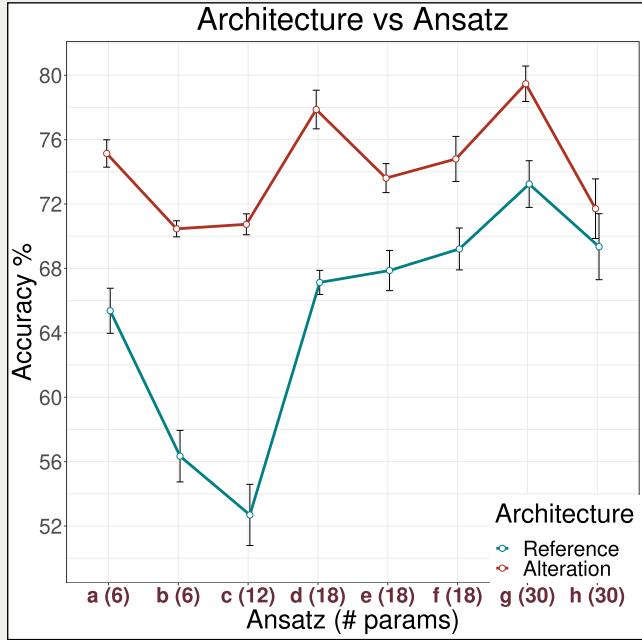
*Reference Architecture:



Lourens, M., Sinayskiy, I., Park, D.K. et al. Hierarchical quantum circuit representations for neural architecture search. npj Quantum Inf 9, 79 (2023).

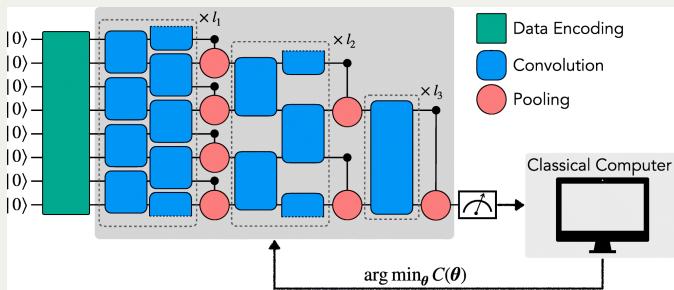
*Hur, T., Kim, L. & Park, D. K. Quantum convolutional neural network for classical data classification. Quantum Mach. Intell. 4, 3 (2022).

Music Genre Classification

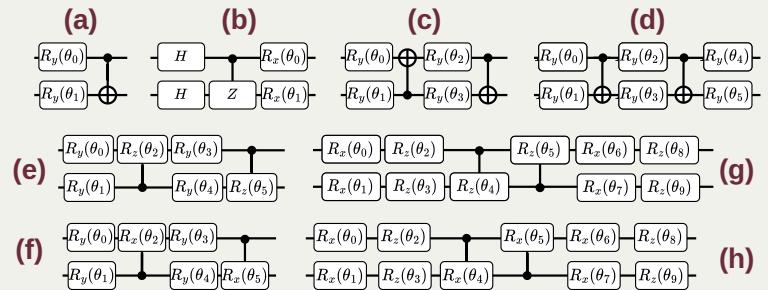


Architecture vs Ansatz				
Ansatz, # Params	Architecture		Δ	Alteration (s_c, F^*, s_p)
	Reference	New alteration		
a, 6	65.37 ± 2.8	75.14 ± 1.7	+9.77	(6, left, 2)
b, 6	56.34 ± 3.2	70.46 ± 1.0	+14.12	(1, odd, 3)
c, 12	52.69 ± 3.8	70.74 ± 1.3	+18.05	(1, odd, 0)
d, 18	67.13 ± 1.5	77.87 ± 2.4	+9.87	(1, outside, 2)
e, 18	67.87 ± 2.5	73.61 ± 1.8	+5.74	(6, left, 0)
f, 18	69.21 ± 2.6	74.80 ± 2.8	+5.59	(1, left, 3)
g, 30	73.24 ± 2.9	79.47 ± 2.2	+6.23	(2, left, 1)
h, 30	69.35 ± 4.1	71.71 ± 3.7	+2.36	(2, left, 1)

*Reference Architecture:



Ansatz:

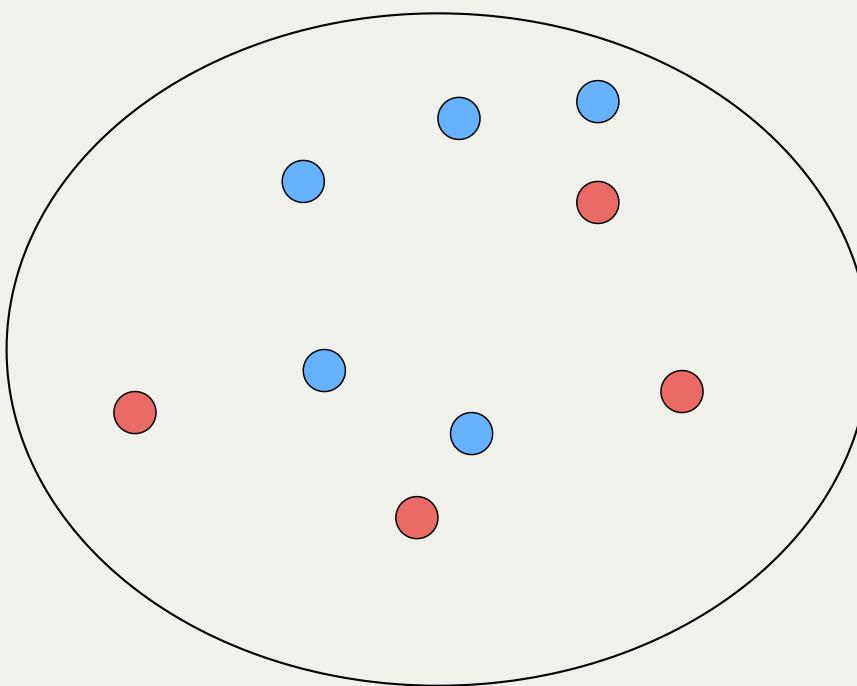


Lourens, M., Sinayskiy, I., Park, D.K. et al. Hierarchical quantum circuit representations for neural architecture search. npj Quantum Inf 9, 79 (2023).

*Hur, T., Kim, L. & Park, D. K. Quantum convolutional neural network for classical data classification. Quantum Mach. Intell. 4, 3 (2022).

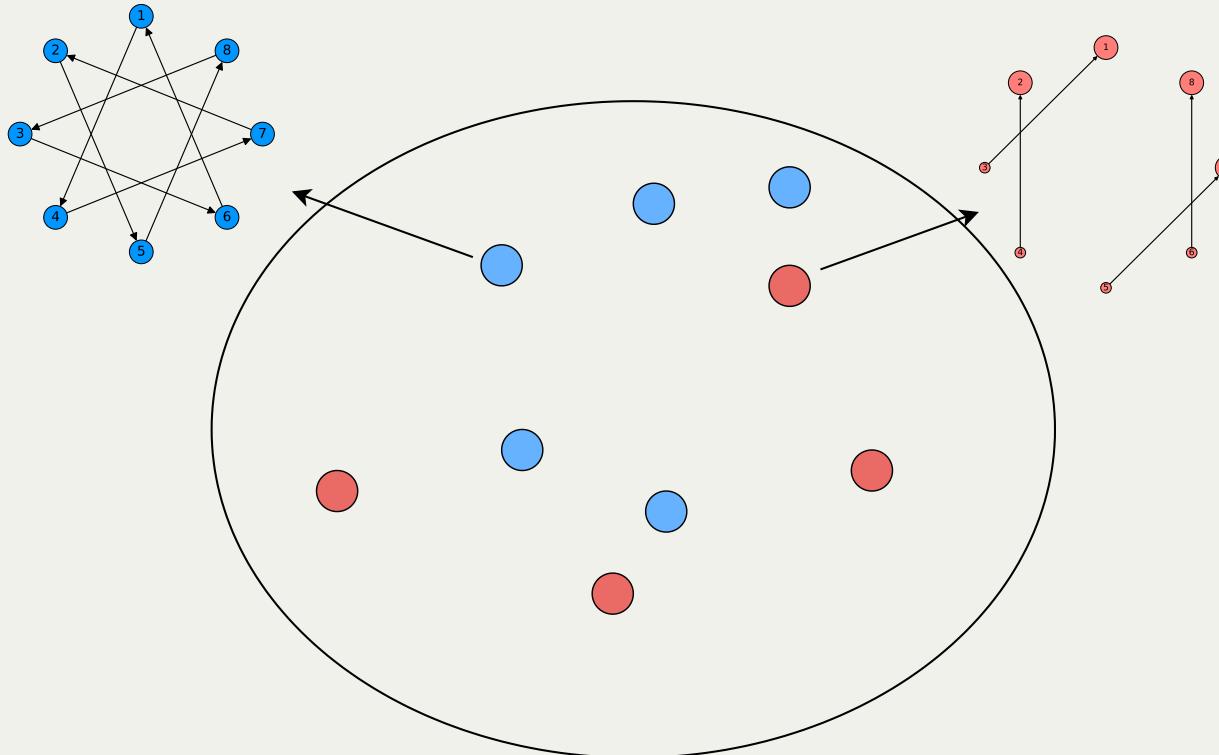
Evolutionary Search

Evolutionary Search



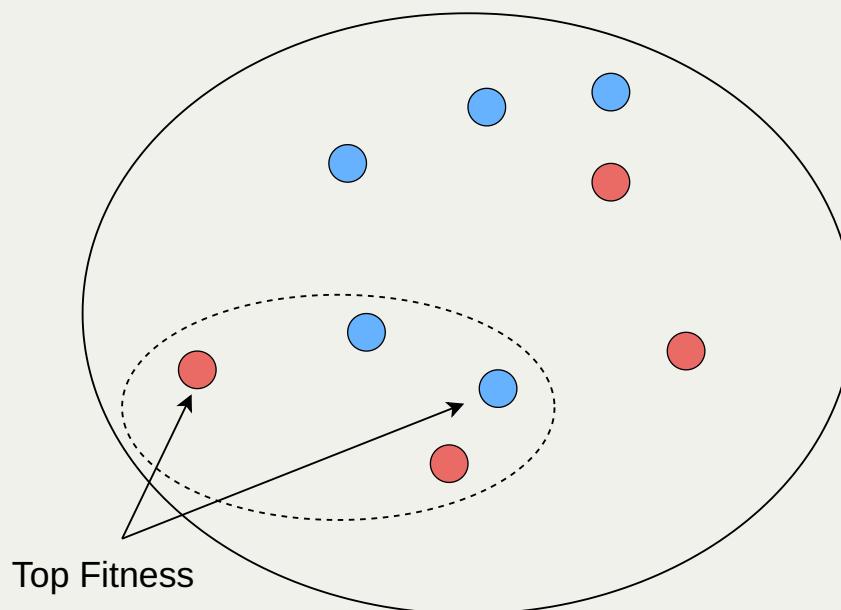
Lourens, M., Sinayskiy, I., Park, D.K. et al. Hierarchical quantum circuit representations for neural architecture search. *npj Quantum Inf* 9, 79 (2023).

Evolutionary Search



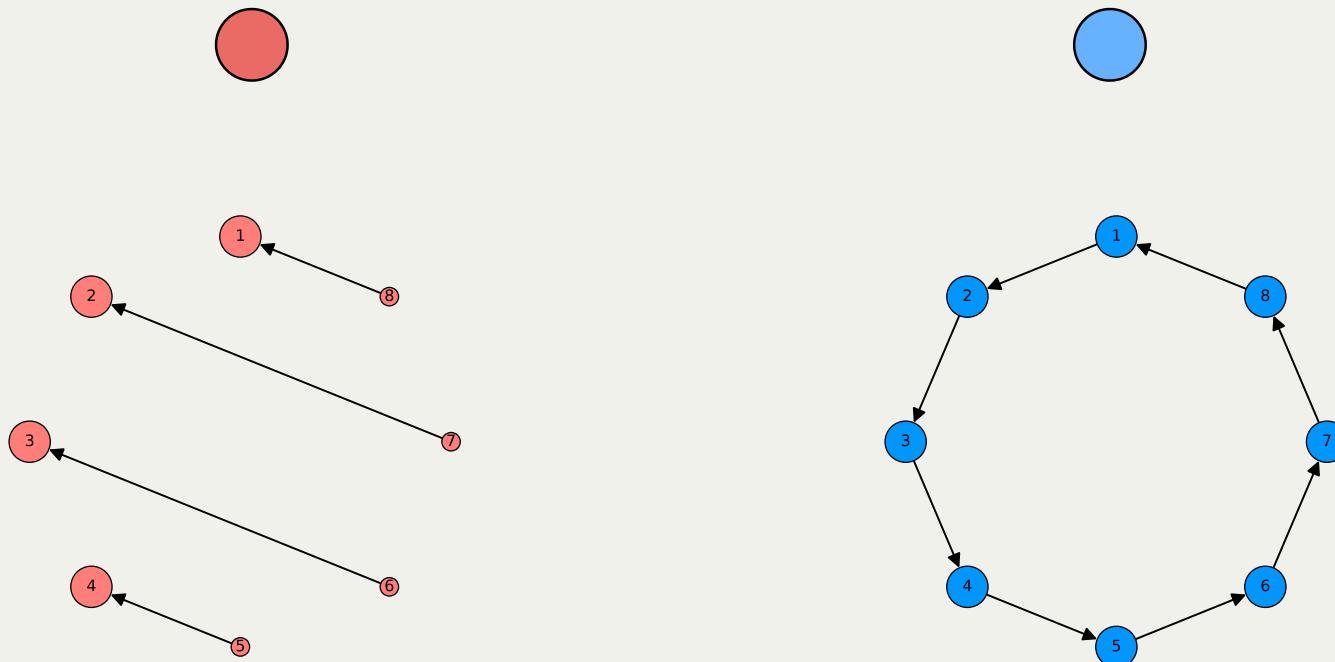
Lourens, M., Sinayskiy, I., Park, D.K. et al. Hierarchical quantum circuit representations for neural architecture search. *npj Quantum Inf* 9, 79 (2023).

Tournament Selection



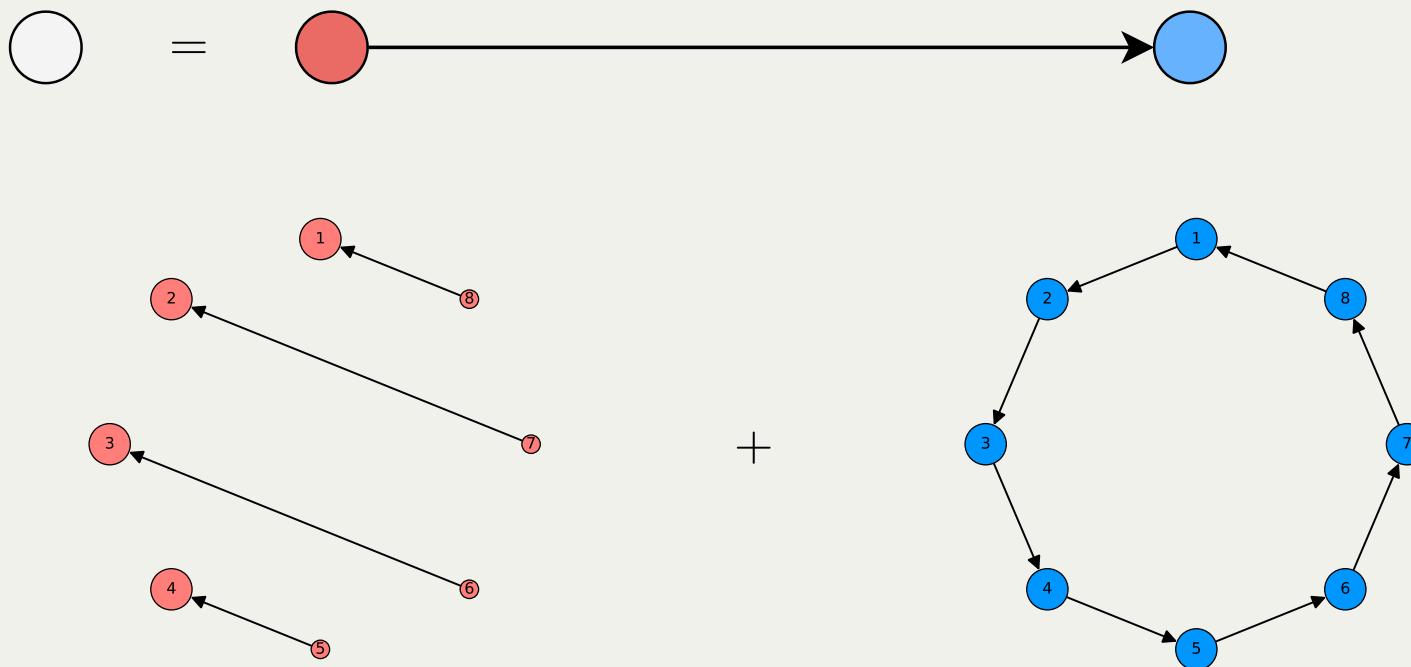
Lourens, M., Sinayskiy, I., Park, D.K. et al. Hierarchical quantum circuit representations for neural architecture search. *npj Quantum Inf* 9, 79 (2023).

Tournament Selection



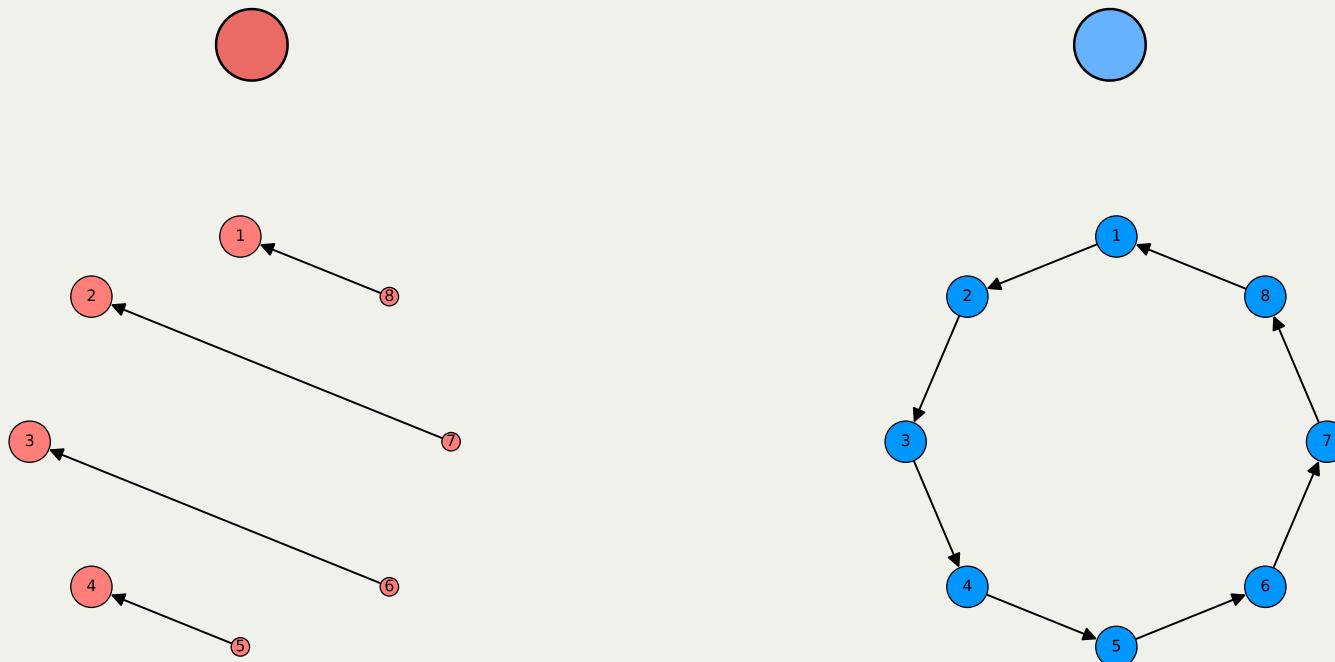
Lourens, M., Sinayskiy, I., Park, D.K. et al. Hierarchical quantum circuit representations for neural architecture search. *npj Quantum Inf* 9, 79 (2023).

Crossover



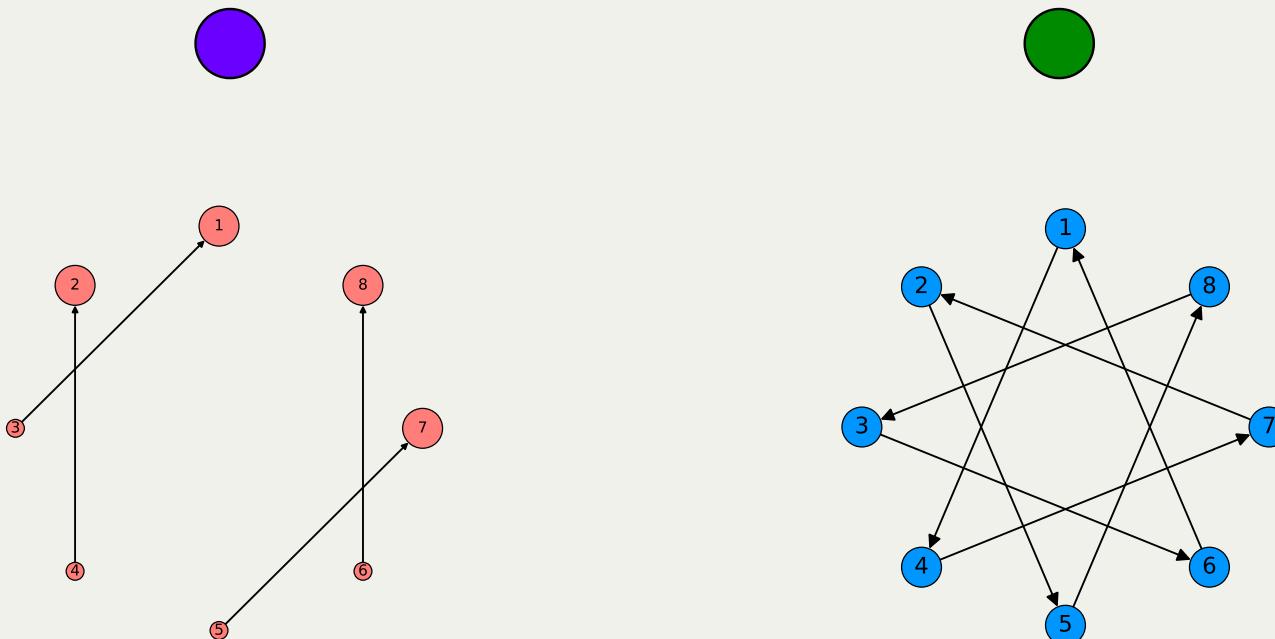
Lourens, M., Sinayskiy, I., Park, D.K. et al. Hierarchical quantum circuit representations for neural architecture search. *npj Quantum Inf* 9, 79 (2023).

Tournament Selection



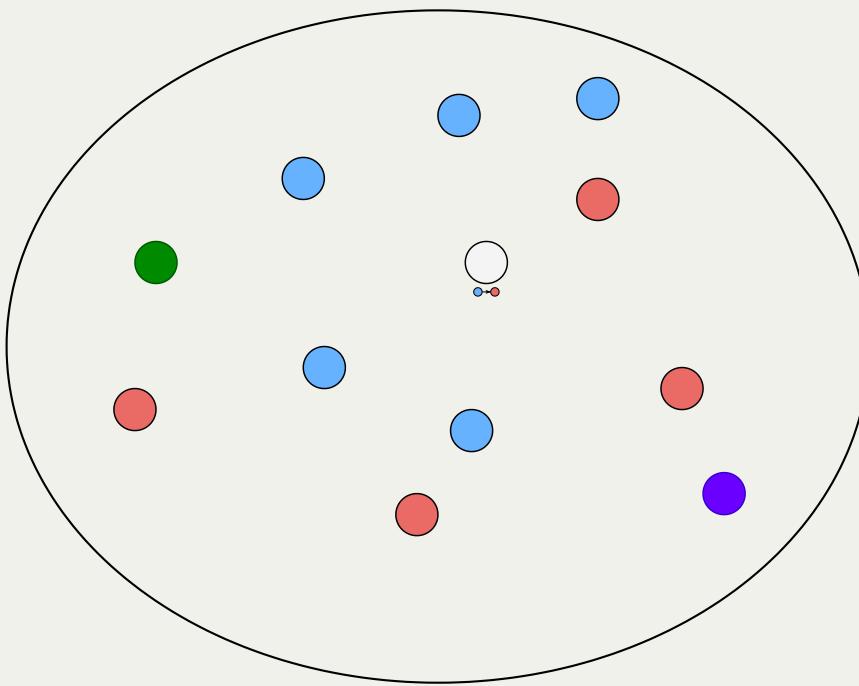
Lourens, M., Sinayskiy, I., Park, D.K. et al. Hierarchical quantum circuit representations for neural architecture search. *npj Quantum Inf* 9, 79 (2023).

Mutation



Lourens, M., Sinayskiy, I., Park, D.K. et al. Hierarchical quantum circuit representations for neural architecture search. *npj Quantum Inf* 9, 79 (2023).

Repeat



Lourens, M., Sinayskiy, I., Park, D.K. et al. Hierarchical quantum circuit representations for neural architecture search. *npj Quantum Inf* 9, 79 (2023).

Quantum Phase Recognition

Cluster-Ising Hamiltonian:

$$H = -J \sum_{i=1}^{N-2} Z_i X_{i+1} Z_{i+2} - h_1 \sum_{i=1}^N X_i - h_2 \sum_{i=1}^{N-1} X_i X_{i+1}.$$

Quantum Phase Recognition

Cluster-Ising Hamiltonian:

$$H = -J \sum_{i=1}^{N-2} Z_i X_{i+1} Z_{i+2} - h_1 \sum_{i=1}^N X_i - h_2 \sum_{i=1}^{N-1} X_i X_{i+1}.$$

- Open boundary Conditions

Quantum Phase Recognition

Cluster-Ising Hamiltonian:

$$H = -J \sum_{i=1}^{N-2} Z_i X_{i+1} Z_{i+2} - h_1 \sum_{i=1}^N X_i - h_2 \sum_{i=1}^{N-1} X_i X_{i+1}.$$

- Open boundary Conditions
- $\mathbb{Z}_2 \times \mathbb{Z}_2$ symmetry-protected topological (SPT) phase P

Quantum Phase Recognition

Cluster-Ising Hamiltonian:

$$H = -J \sum_{i=1}^{N-2} Z_i X_{i+1} Z_{i+2} - h_1 \sum_{i=1}^N X_i - h_2 \sum_{i=1}^{N-1} X_i X_{i+1}.$$

- Open boundary Conditions
- $\mathbb{Z}_2 \times \mathbb{Z}_2$ symmetry-protected topological (SPT) phase P
 - $X_{\text{even(odd)}} = \prod_{i \in \text{even(odd)}} X_i$

Quantum Phase Recognition

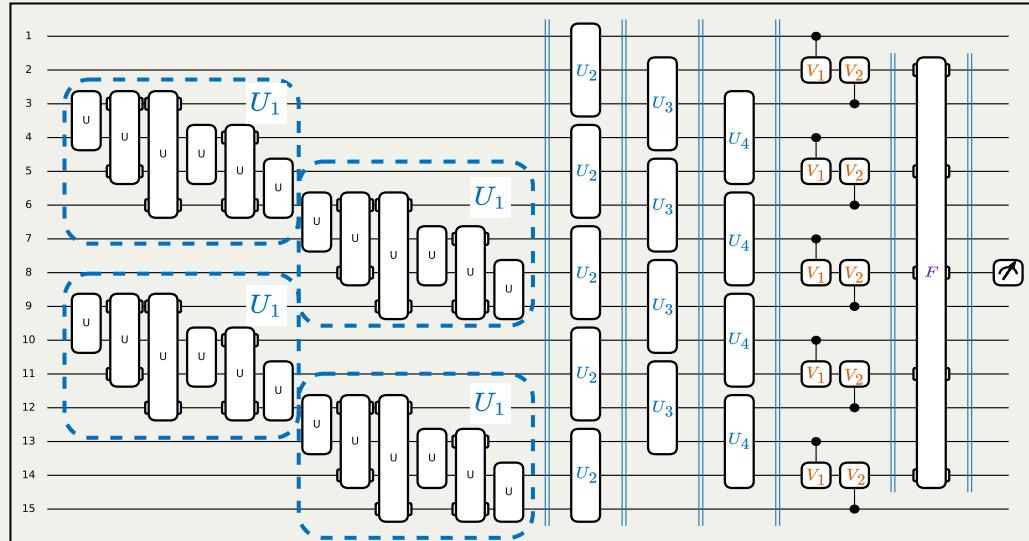
Cluster-Ising Hamiltonian:

$$H = -J \sum_{i=1}^{N-2} Z_i X_{i+1} Z_{i+2} - h_1 \sum_{i=1}^N X_i - h_2 \sum_{i=1}^{N-1} X_i X_{i+1}.$$

- Open boundary Conditions
- $\mathbb{Z}_2 \times \mathbb{Z}_2$ symmetry-protected topological (SPT) phase P
 - $X_{\text{even(odd)}} = \prod_{i \in \text{even(odd)}} X_i$
- Given an unknown ground state $|\psi_g\rangle$, what is the phase?

Quantum Phase Recognition

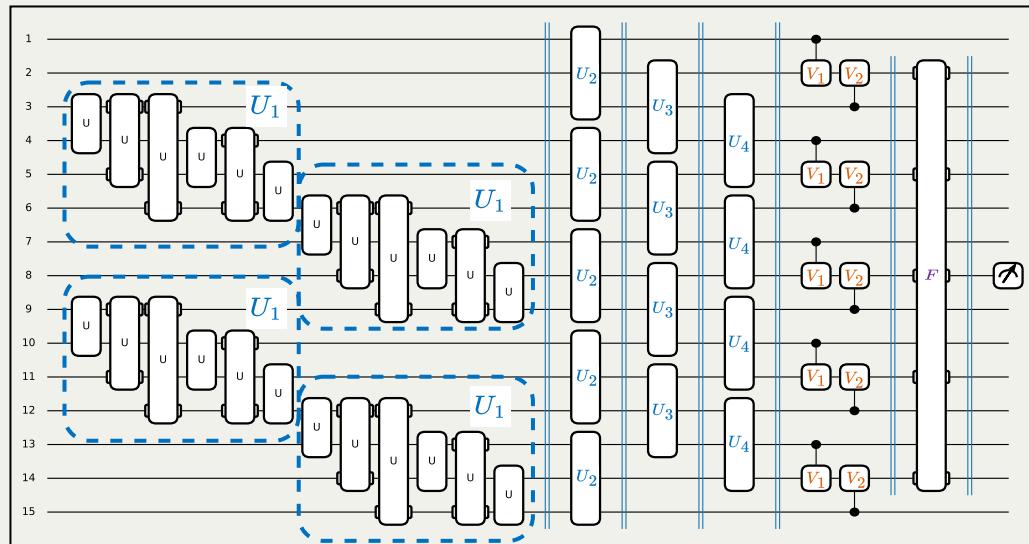
```
1 u0 = Qinit(4) + Qpermute(share_weights=False, mapping=Ugm)
2 u1 = Qcycle(1,3,2, mapping=u0, boundary="open", edge_order=[1,3,2,4])
3 u2 = sum([Qcycle(1,3,offset, mapping=Ugm, boundary="open") for offset in range(3)])
4 u3 = Qmask("101",1,3,0, mapping=Vgm, boundary="open")
5 u4 = Qcycle(1, mapping=Ugm, boundary="open")
6 motif = Qinit(15) + u1 + u2 + u3 + u4
```



Lourens, M., Sinayskiy, I., Park, D.K. et al. Hierarchical quantum circuit representations for neural architecture search. npj Quantum Inf 9, 79 (2023).

Quantum Phase Recognition

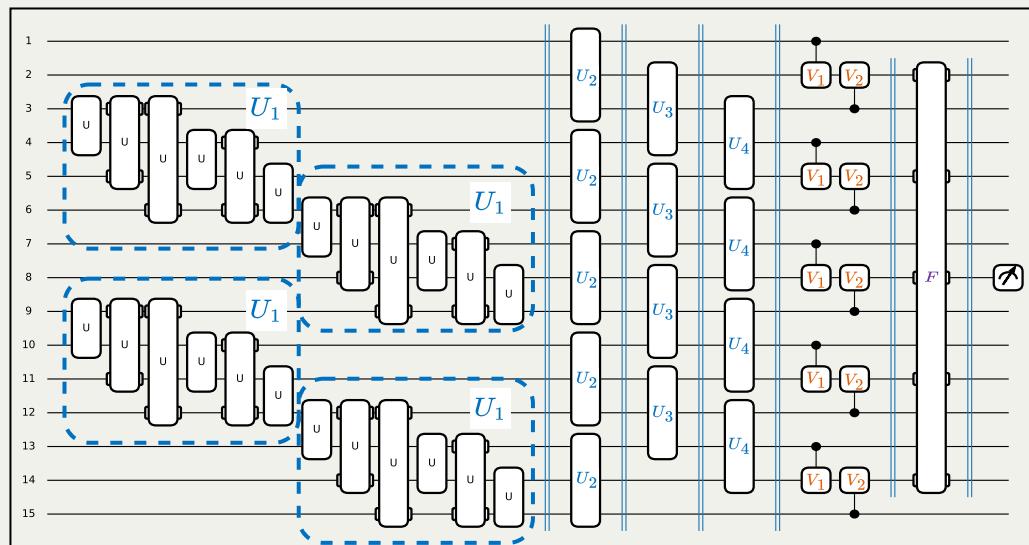
```
1 u0 = Qinit(4) + Qpermute(share_weights=False, mapping=Ugm)
2 u1 = Qcycle(1,3,2, mapping=u0, boundary="open", edge_order=[1,3,2,4])
3 u2 = sum([Qcycle(1,3,offset, mapping=Ugm, boundary="open") for offset in range(3)])
4 u3 = Qmask("101",1,3,0, mapping=Vgm, boundary="open")
5 u4 = Qcycle(1, mapping=Ugm, boundary="open")
6 motif = Qinit(15) + u1 + u2 + u3 + u4
```



Lourens, M., Sinayskiy, I., Park, D.K. et al. Hierarchical quantum circuit representations for neural architecture search. npj Quantum Inf 9, 79 (2023).

Quantum Phase Recognition

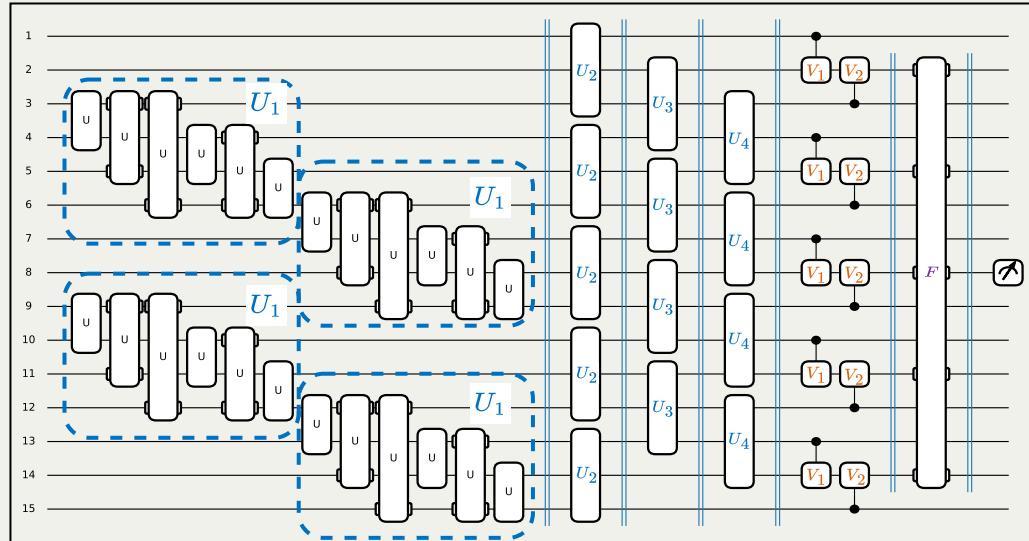
```
1 u0 = Qinit(4) + Qpermute(share_weights=False, mapping=Ugm)
2 u1 = Qcycle(1,3,2, mapping=u0, boundary="open", edge_order=[1,3,2,4])
3 u2 = sum([Qcycle(1,3,offset, mapping=Ugm, boundary="open") for offset in range(3)])
4 u3 = Qmask("101",1,3,0, mapping=Vgm, boundary="open")
5 u4 = Qcycle(1, mapping=Ugm, boundary="open")
6 motif = Qinit(15) + u1 + u2 + u3 + u4
```



Lourens, M., Sinayskiy, I., Park, D.K. et al. Hierarchical quantum circuit representations for neural architecture search. npj Quantum Inf 9, 79 (2023).

Quantum Phase Recognition

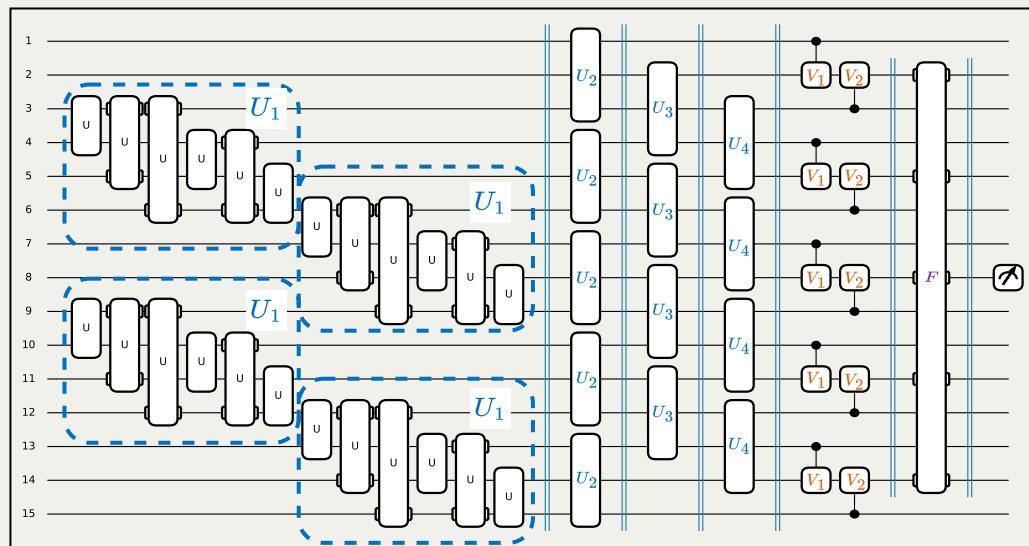
```
1 u0 = Qinit(4) + Qpermute(share_weights=False, mapping=Ugm)
2 u1 = Qcycle(1,3,2, mapping=u0, boundary="open", edge_order=[1,3,2,4])
3 u2 = sum([Qcycle(1,3,offset, mapping=Ugm, boundary="open") for offset in range(3)])
4 u3 = Qmask("101",1,3,0, mapping=Vgm, boundary="open")
5 u4 = Qcycle(1, mapping=Ugm, boundary="open")
6 motif = Qinit(15) + u1 + u2 + u3 + u4
```



Lourens, M., Sinayskiy, I., Park, D.K. et al. Hierarchical quantum circuit representations for neural architecture search. npj Quantum Inf 9, 79 (2023).

Quantum Phase Recognition

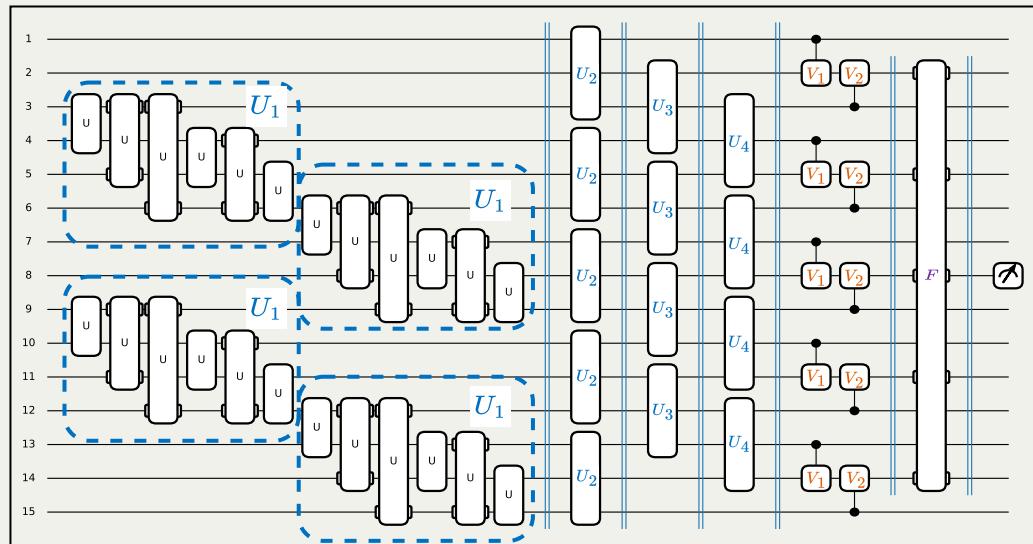
```
1 u0 = Qinit(4) + Qpermute(share_weights=False, mapping=Ugm)
2 u1 = Qcycle(1,3,2, mapping=u0, boundary="open", edge_order=[1,3,2,4])
3 u2 = sum([Qcycle(1,3,offset, mapping=Ugm, boundary="open") for offset in range(3)])
4 u3 = Qmask("101",1,3,0, mapping=Vgm, boundary="open")
5 u4 = Qcycle(1, mapping=Ugm, boundary="open")
6 motif = Qinit(15) + u1 + u2 + u3 + u4
```



Lourens, M., Sinayskiy, I., Park, D.K. et al. Hierarchical quantum circuit representations for neural architecture search. npj Quantum Inf 9, 79 (2023).

Quantum Phase Recognition

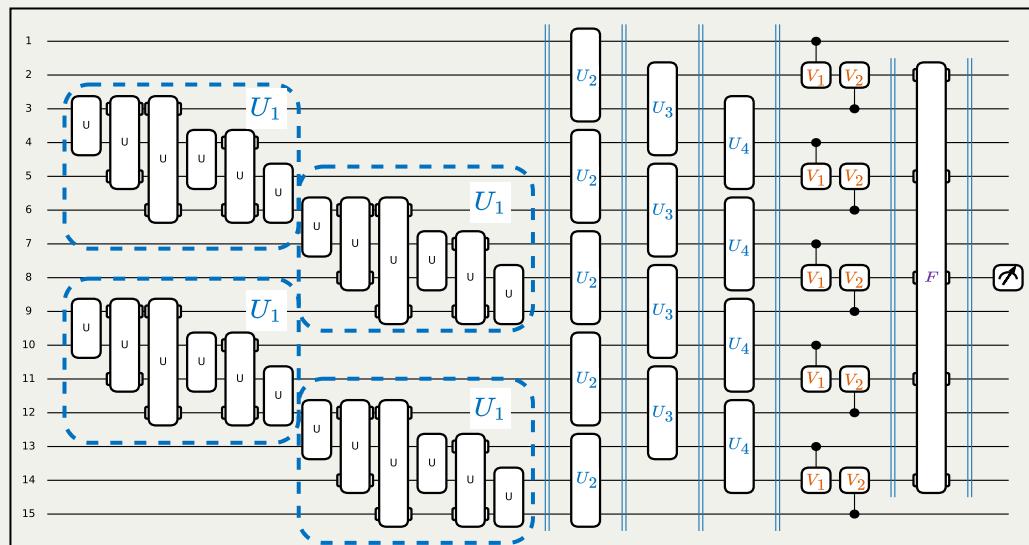
```
1 u0 = Qinit(4) + Qpermute(share_weights=False, mapping=Ugm)
2 u1 = Qcycle(1,3,2, mapping=u0, boundary="open", edge_order=[1,3,2,4])
3 u2 = sum([Qcycle(1,3,offset, mapping=Ugm, boundary="open") for offset in range(3)])
4 u3 = Qmask("101",1,3,0, mapping=Vgm, boundary="open")
5 u4 = Qcycle(1, mapping=Ugm, boundary="open")
6 motif = Qinit(15) + u1 + u2 + u3 + u4
```



Lourens, M., Sinayskiy, I., Park, D.K. et al. Hierarchical quantum circuit representations for neural architecture search. npj Quantum Inf 9, 79 (2023).

Quantum Phase Recognition

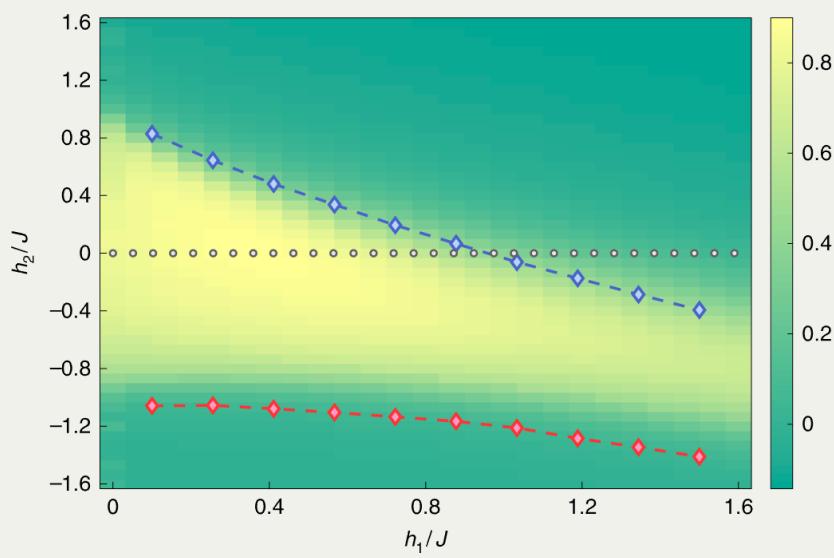
```
1 u0 = Qinit(4) + Qpermute(share_weights=False, mapping=Ugm)
2 u1 = Qcycle(1,3,2, mapping=u0, boundary="open", edge_order=[1,3,2,4])
3 u2 = sum([Qcycle(1,3,offset, mapping=Ugm, boundary="open") for offset in range(3)])
4 u3 = Qmask("101",1,3,0, mapping=Vgm, boundary="open")
5 u4 = Qcycle(1, mapping=Ugm, boundary="open")
6 motif = Qinit(15) + u1 + u2 + u3 + u4
```



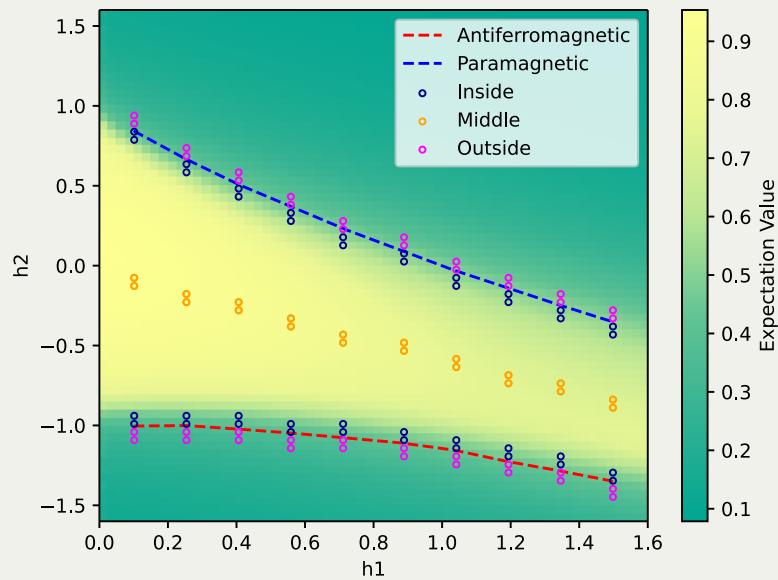
Lourens, M., Sinayskiy, I., Park, D.K. et al. Hierarchical quantum circuit representations for neural architecture search. npj Quantum Inf 9, 79 (2023).

Quantum Phase Recognition

Reference¹



Found²

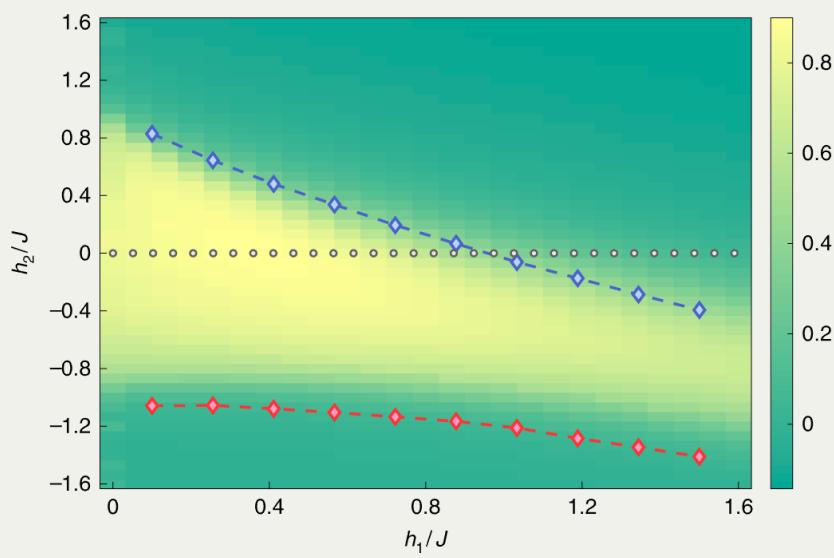


¹Cong, I., Choi, S. & Lukin, M.D. Quantum convolutional neural networks. Nat. Phys. 15, 1273-1278 (2019).

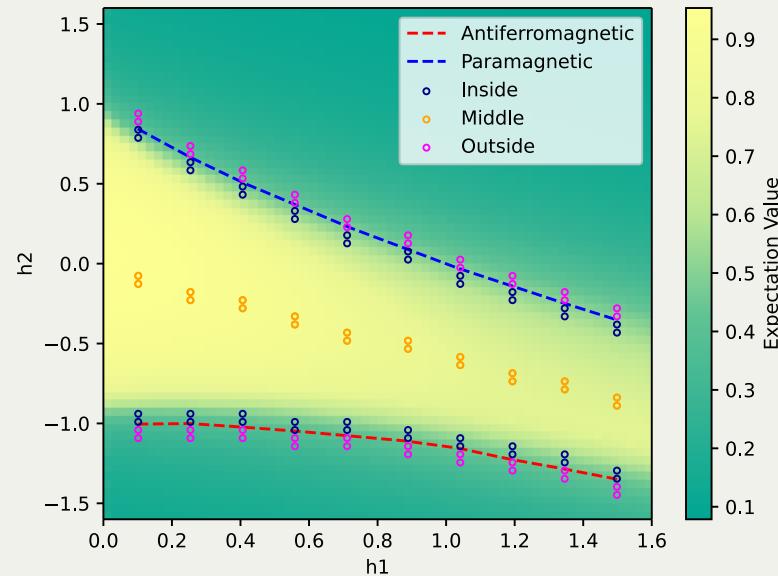
²Lourens, M., Sinayskiy, I., Park, D.K. et al. Hierarchical quantum circuit representations for neural architecture search. npj Quantum Inf 9, 79 (2023).

Quantum Phase Recognition

Reference¹



Found²



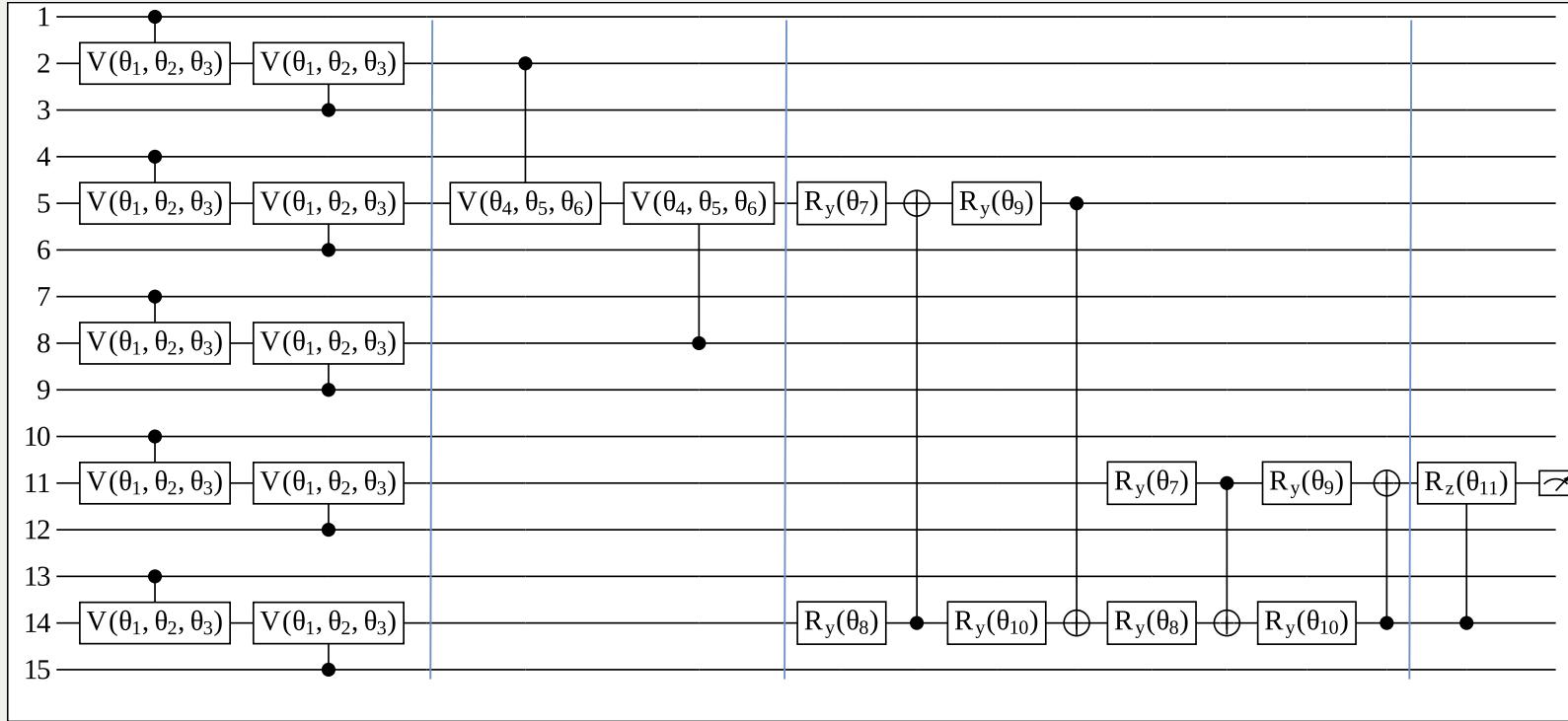
Metric	Reference	Found
Number of parameters	1308	11
Sample Complexity (Inside)	61.523	36.079
Sample Complexity (Middle)	10.992	13.253
MSE (Outside)	0.164	0.167

¹Cong, I., Choi, S. & Lukin, M.D. Quantum convolutional neural networks. Nat. Phys. 15, 1273-1278 (2019).

²Lourens, M., Sinayskiy, I., Park, D.K. et al. Hierarchical quantum circuit representations for neural architecture search. npj Quantum Inf 9, 79 (2023).

Quantum Phase Recognition

Found:



Metric	Reference	Found
Number of parameters	1308	11
Sample Complexity (Inside)	61.523	36.079
Sample Complexity (Middle)	10.992	13.253
MSE (Outside)	0.164	0.167

Programming

Shor's Algorithm

- Modular Exponentiation
- Quantum Fourier Transform

Vedral, V., Barenco, A. & Ekert, A. Quantum networks for elementary arithmetic operations. Phys. Rev. A 54, 147-153 (1996)

Shor's Algorithm

- Modular Exponentiation
 - Ctrl-Mult modulo N
- Quantum Fourier Transform

Vedral, V., Barenco, A. & Ekert, A. Quantum networks for elementary arithmetic operations. Phys. Rev. A 54, 147-153 (1996)

Shor's Algorithm

- Modular Exponentiation
 - Ctrl-Mult modulo N
 - Addition modulo N
- Quantum Fourier Transform

Vedral, V., Barenco, A. & Ekert, A. Quantum networks for elementary arithmetic operations. Phys. Rev. A 54, 147-153 (1996)

Shor's Algorithm

- Modular Exponentiation
 - Ctrl-Mult modulo N
 - Addition modulo N
 - Addition
 - Quantum Fourier Transform

Vedral, V., Barenco, A. & Ekert, A. Quantum networks for elementary arithmetic operations. Phys. Rev. A 54, 147-153 (1996)

Shor's Algorithm

- Modular Exponentiation
 - Ctrl-Mult modulo N
 - Addition modulo N
 - Addition
 - Carry
 - xor
- Quantum Fourier Transform

Vedral, V., Barenco, A. & Ekert, A. Quantum networks for elementary arithmetic operations. Phys. Rev. A 54, 147-153 (1996)

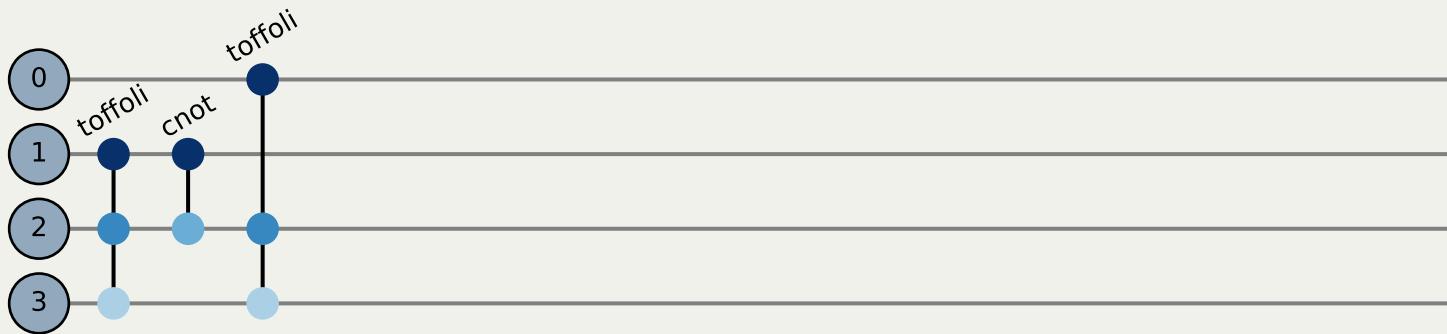
Shor's Algorithm

- Modular Exponentiation
 - Ctrl-Mult modulo N
 - Addition modulo N
 - Addition
 - Carry
 - cnot
 - Toffoli
 - xor
 - cnot
 - Quantum Fourier Transform

Vedral, V., Barenco, A. & Ekert, A. Quantum networks for elementary arithmetic operations. Phys. Rev. A 54, 147-153 (1996)

Shor circuit

```
1 # ===== Motifs level 1
2 carry_motif = (
3     Qinit(4)
4     + Qmotif(E=[(1, 2, 3)], mapping=toffoli)
5     + Qmotif(E=[(1, 2)], mapping=cnot)
6     + Qmotif(E=[(0, 2, 3)], mapping=toffoli)
7 )
8
9 # Turn carry into a function to be able to reverse it easily
10 carry = lambda r: carry_motif if r == 1 else carry_motif.reverse()
11 plot_circuit(carry(1))
12 plot_circuit(carry(-1))
```



Shor circuit

```
1 # ===== Motifs level 1
2 carry_motif = (
3     Qinit(4)
4     + Qmotif(E=[(1, 2, 3)], mapping=toffoli)
5     + Qmotif(E=[(1, 2)], mapping=cnot)
6     + Qmotif(E=[(0, 2, 3)], mapping=toffoli)
7 )
8
9 # Turn carry into a function to be able to reverse it easily
10 carry = lambda r: carry_motif if r == 1 else carry_motif.reverse()
11 plot_circuit(carry(1))
12 plot_circuit(carry(-1))
```



Shor circuit

```
1 # ===== Motifs level 1
2 sum = lambda r=1: Qinit(3, name=f"sum") + Qpivot(
3     "*1", merge_within="01", mapping=cnot, edge_order=[-1 * r]
4 )
5
6 plot_circuit(sum(1))
7 plot_circuit(sum(-1))
```



<https://github.com/matt-lourens/hierarqcal>

Shor circuit

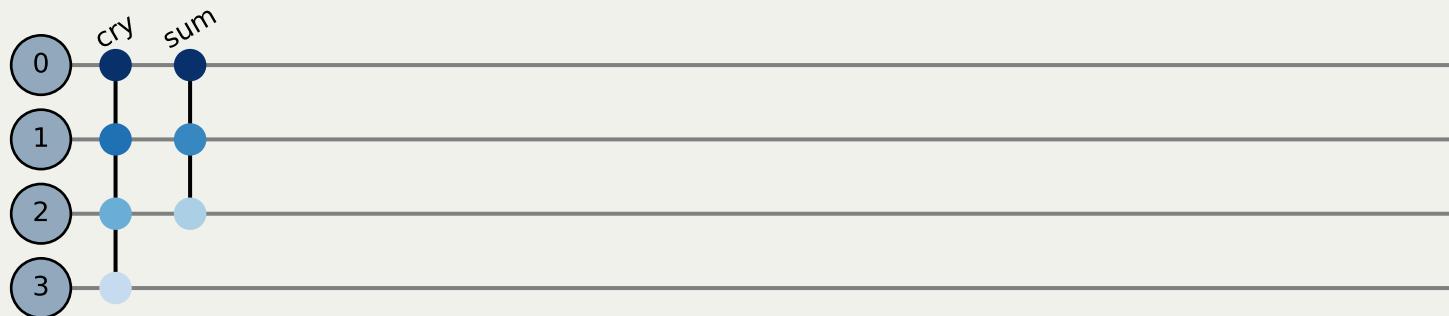
```
1 # ===== Motifs level 1
2 sum = lambda r=1: Qinit(3, name=f"sum") + Qpivot(
3     "*1", merge_within="01", mapping=cnot, edge_order=[-1 * r]
4 )
5
6 plot_circuit(sum(1))
7 plot_circuit(sum(-1))
```



<https://github.com/matt-lourens/hierarqcal>

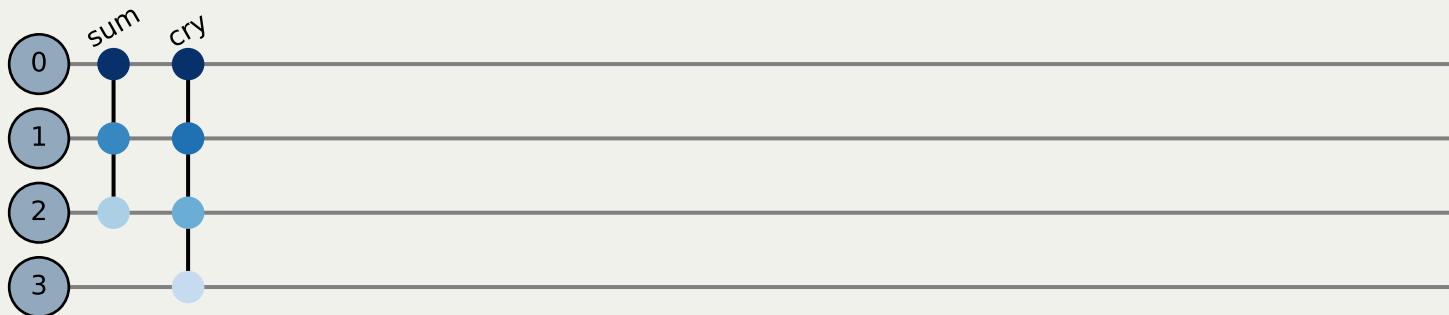
Shor circuit

```
1 # ===== Motifs level 2
2 carry_sum_motif = (
3     lambda r=1: Qinit(4, name=f"crs")
4     + Qpivot("1*", merge_within="1111", mapping=carry(-r))
5     + Qpivot("1*", merge_within="111", mapping=sum(r))
6 )
7
8 carry_sum = lambda r=1: carry_sum_motif(1) if r == 1 else carry_sum_motif(-1).reverse
9
10 plot_circuit(carry_sum(1))
11 plot_circuit(carry_sum(-1))
```



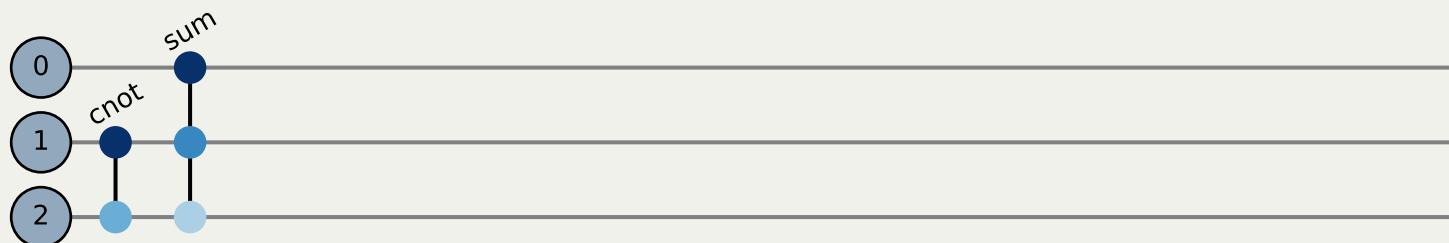
Shor circuit

```
1 # ===== Motifs level 2
2 carry_sum_motif = (
3     lambda r=1: Qinit(4, name=f"crs")
4     + Qpivot("1*", merge_within="1111", mapping=carry(-r))
5     + Qpivot("1*", merge_within="111", mapping=sum(r))
6 )
7
8 carry_sum = lambda r=1: carry_sum_motif(1) if r == 1 else carry_sum_motif(-1).reverse
9
10 plot_circuit(carry_sum(1))
11 plot_circuit(carry_sum(-1))
```



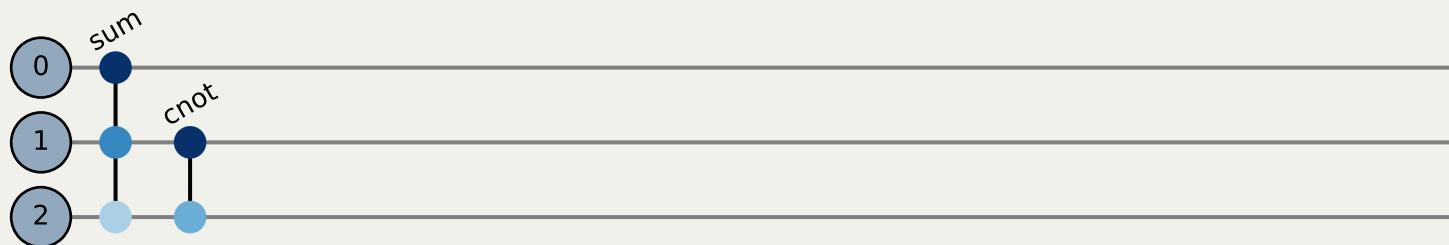
Shor circuit

```
1 cnot_sum_motif = (
2     lambda r=1: Qinit(3, name=f"cns")
3     + Qpivot("*1", merge_within="11", mapping=cnot)
4     + Qpivot("*1", merge_within="111", mapping=sum(r))
5 )
6
7 cnot_sum = lambda r=1: cnot_sum_motif(1) if r == 1 else cnot_sum_motif(-1).reverse()
8
9 plot_circuit(cnot_sum(1))
10 plot_circuit(cnot_sum(-1))
```



Shor circuit

```
1 cnot_sum_motif = (
2     lambda r=1: Qinit(3, name=f"cns")
3     + Qpivot("*1", merge_within="11", mapping=cnot)
4     + Qpivot("*1", merge_within="111", mapping=sum(r))
5 )
6
7 cnot_sum = lambda r=1: cnot_sum_motif(1) if r == 1 else cnot_sum_motif(-1).reverse()
8
9 plot_circuit(cnot_sum(1))
10 plot_circuit(cnot_sum(-1))
```



Shor circuit

```
1 # ===== Motifs level 3
2 carry_layer = lambda r=1: Qcycle(
3     1,
4     3,
5     0,
6     mapping=carry(r),
7     boundary="open",
8     edge_order=[r],
9 )
10 plot_circuit(carry_layer_1)
```



Shor circuit

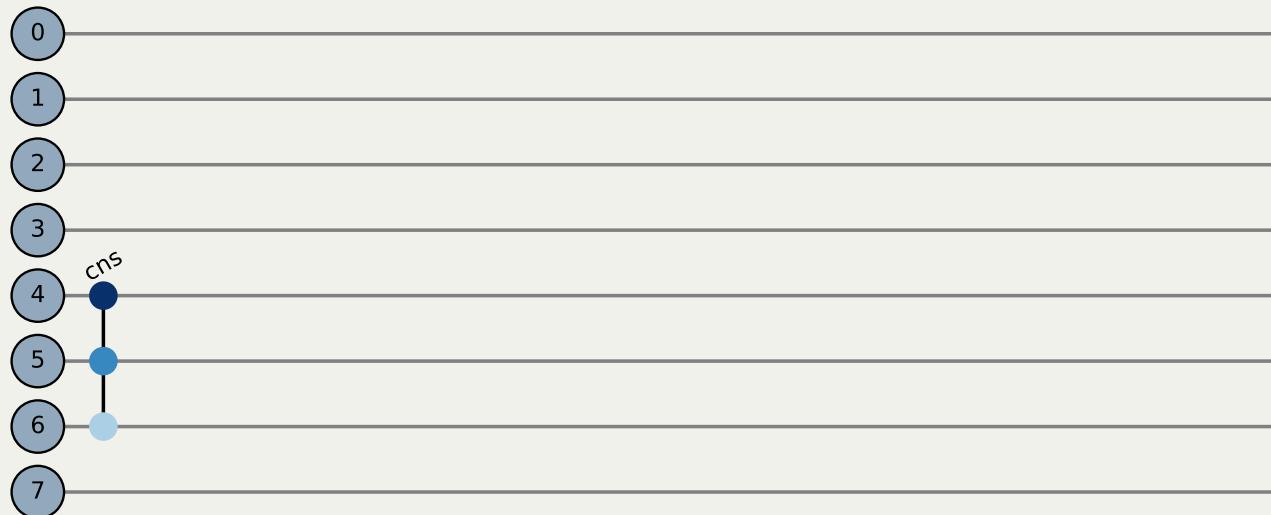
```
1 cnot_sum_pivot = lambda r: Qpivot("*10", merge_within="111", mapping=cnot_sum(r))
2
3 plot_circuit(Qinit(6) + cnot_sum_pivot(1))
4 plot_circuit(Qinit(8) + cnot_sum_pivot(1))
```



<https://github.com/matt-lourens/hierarqcal>

Shor circuit

```
1 cnot_sum_pivot = lambda r: Qpivot("*10", merge_within="111", mapping=cnot_sum(r))
2
3 plot_circuit(Qinit(6) + cnot_sum_pivot(1))
4 plot_circuit(Qinit(8) + cnot_sum_pivot(1))
```



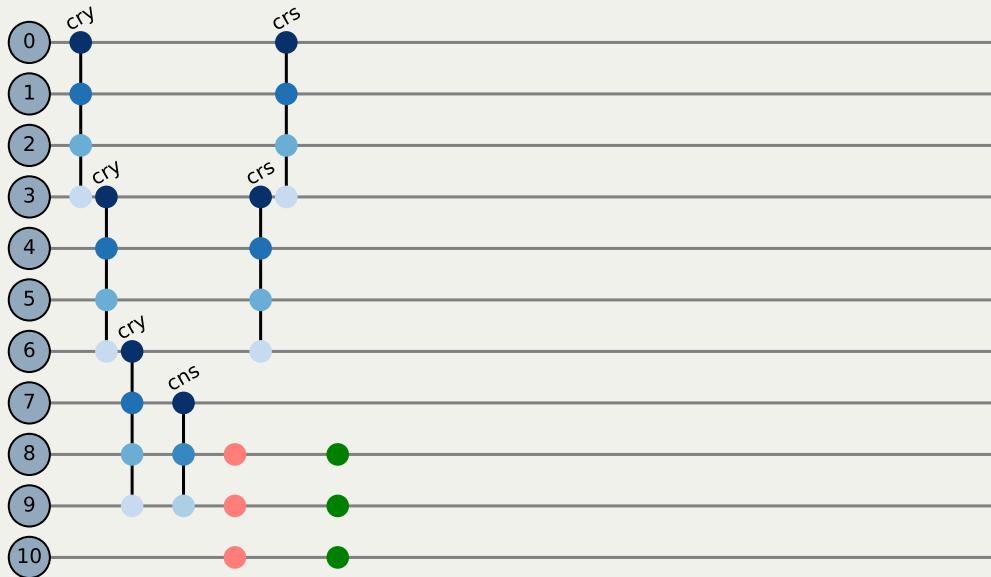
Shor circuit

```
1 carry_sum_layer = lambda r: (
2     Qmask("*111")
3     + Qcycle(1, 3, 0,
4             boundary="open",
5             mapping=carry_sum(r),
6             edge_order=[-r],
7         )
8     + Qunmask("previous")
9 )
10 plot_circuit(Qinit(10) + carry_sum_layer(1))
```



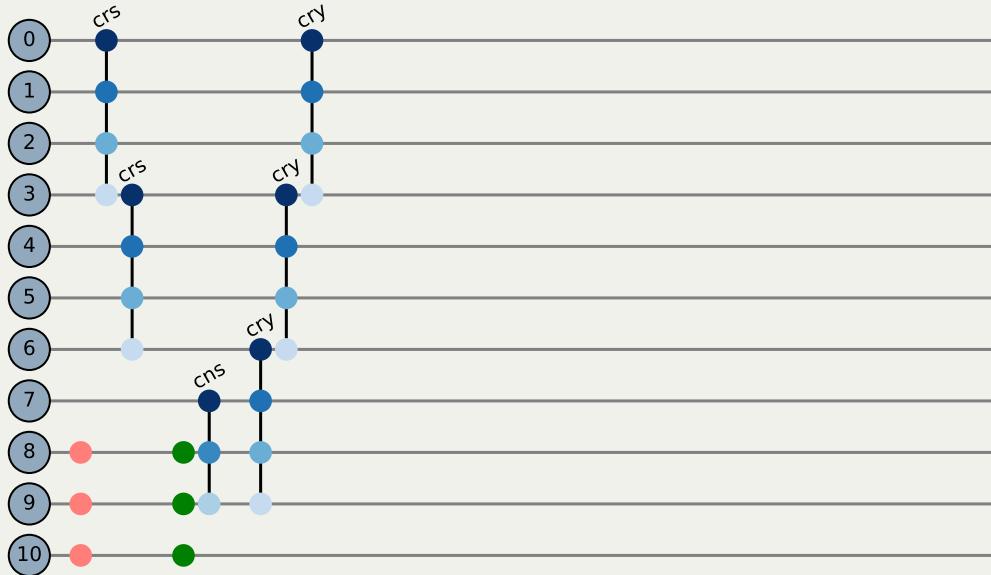
Shor circuit

```
1 addition = carry_layer(1) + cnot_sum_pivot(1) + carry_sum_layer(1)
2 subtraction = carry_sum_layer(-1) + cnot_sum_pivot(-1) + carry_layer(-1)
3
4 plot_circuit(Qinit(11) + addition)
5 plot_circuit(Qinit(11) + subtraction)
6 plot_circuit(Qinit(13) + addition)
7 plot_circuit(Qinit(13) + subtraction)
```



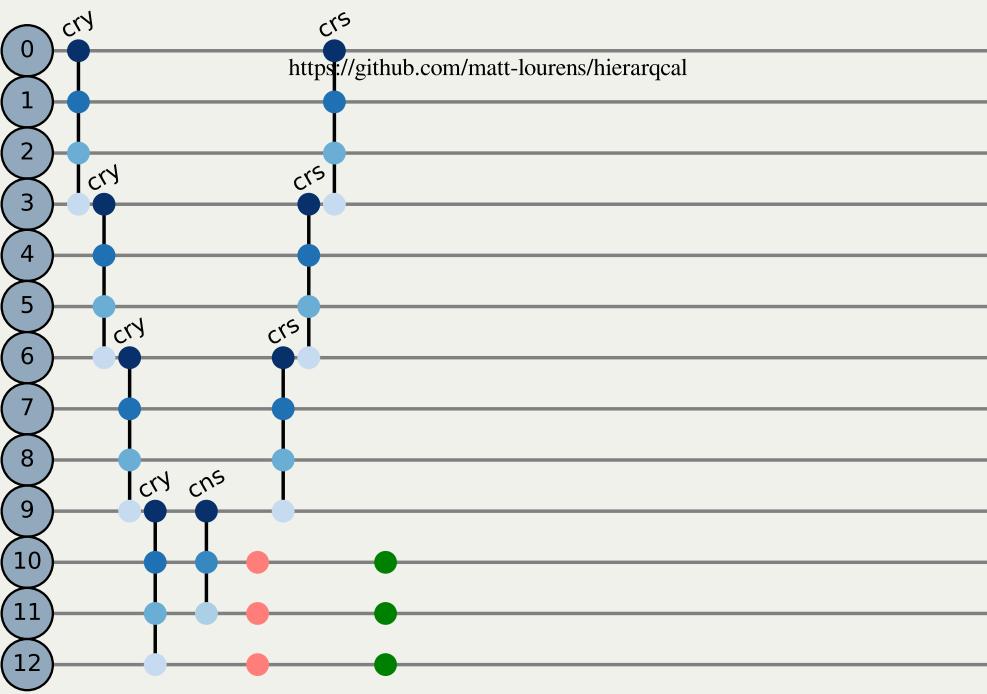
Shor circuit

```
1 addition = carry_layer(1) + cnot_sum_pivot(1) + carry_sum_layer(1)
2 subtraction = carry_sum_layer(-1) + cnot_sum_pivot(-1) + carry_layer(-1)
3
4 plot_circuit(Qinit(11) + addition)
5 plot_circuit(Qinit(11) + subtraction)
6 plot_circuit(Qinit(13) + addition)
7 plot_circuit(Qinit(13) + subtraction)
```



Shor circuit

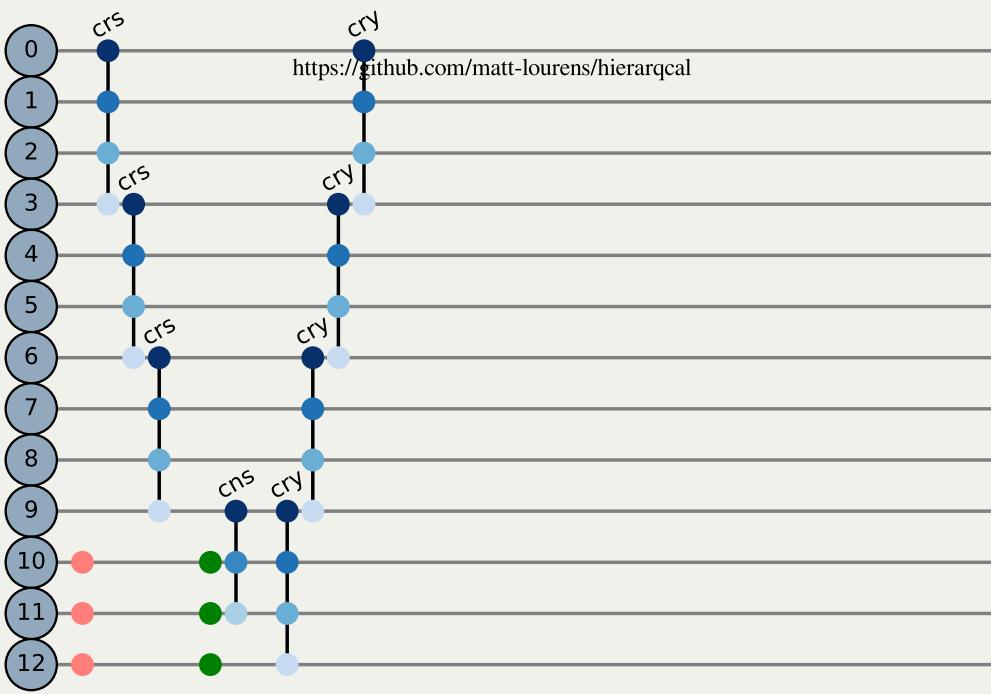
```
1 addition = carry_layer(1) + cnot_sum_pivot(1) + carry_sum_layer(1)
2 subtraction = carry_sum_layer(-1) + cnot_sum_pivot(-1) + carry_layer(-1)
3
4 plot_circuit(Qinit(11) + addition)
5 plot_circuit(Qinit(11) + subtraction)
6 plot_circuit(Qinit(13) + addition)
7 plot_circuit(Qinit(13) + subtraction)
```



<https://github.com/matt-lourens/hierarqcal>

Shor circuit

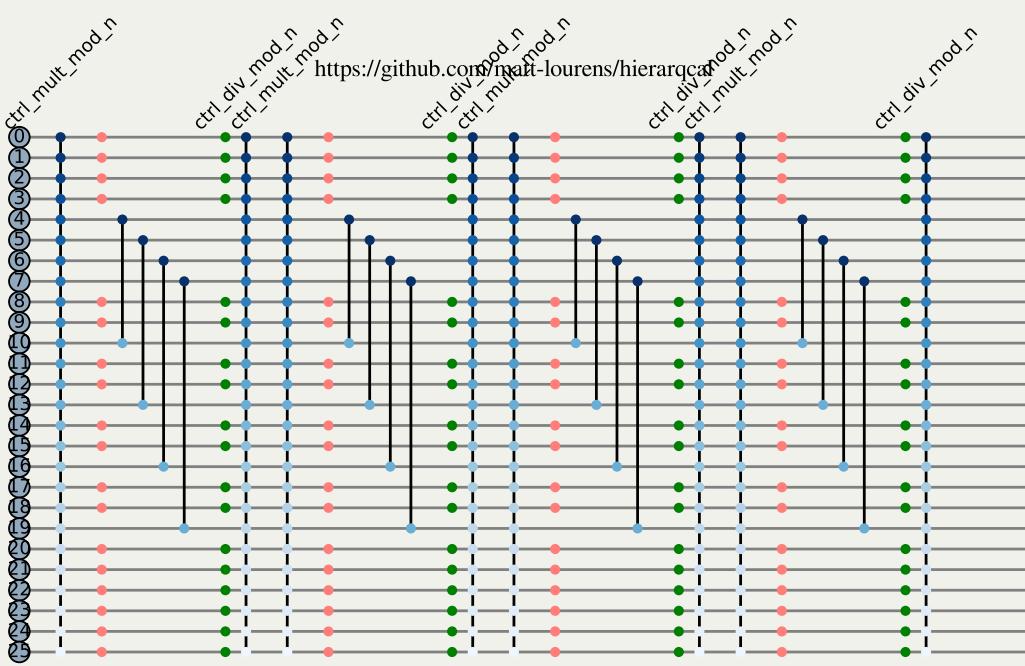
```
1 addition = carry_layer(1) + cnot_sum_pivot(1) + carry_sum_layer(1)
2 subtraction = carry_sum_layer(-1) + cnot_sum_pivot(-1) + carry_layer(-1)
3
4 plot_circuit(Qinit(11) + addition)
5 plot_circuit(Qinit(11) + subtraction)
6 plot_circuit(Qinit(13) + addition)
7 plot_circuit(Qinit(13) + subtraction)
```



<https://github.com/matt-lourens/hierarqcal>

Shor circuit

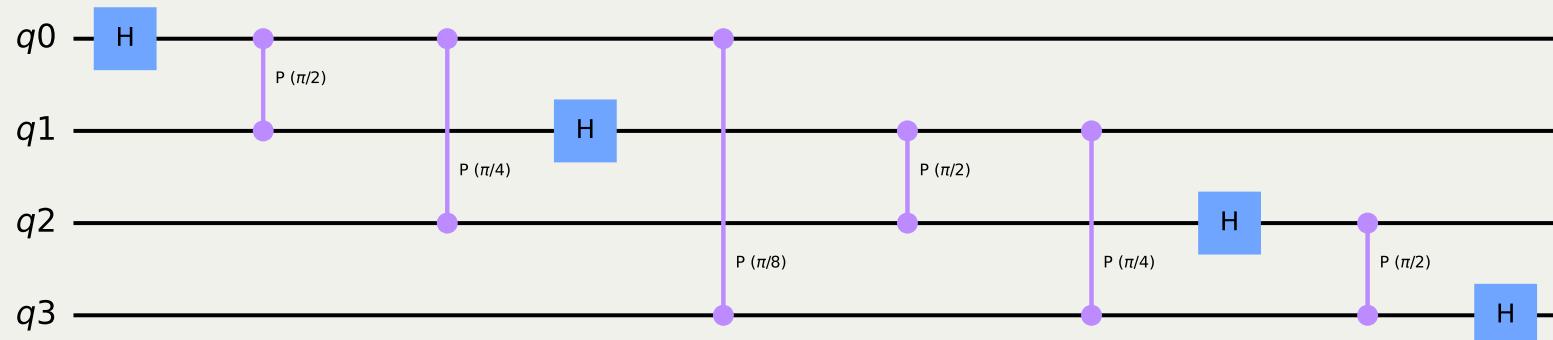
```
1 exp_mod_n = tuple()
2 for k in range(n):
3     exp_mod_n += (
4         Qpivot(mapping=ctrl_mult(a ** (2**k), N, n, ctrl=k, divide=False))
5         + swap_bx
6         + Qpivot(mapping=ctrl_mult(a ** (2**k), N, n, ctrl=k, divide=True)))
7     )
8
9 hierq = Qinit(nq, tensors=tensors) + exp_mod_n
```



Quantum Fourier Transform

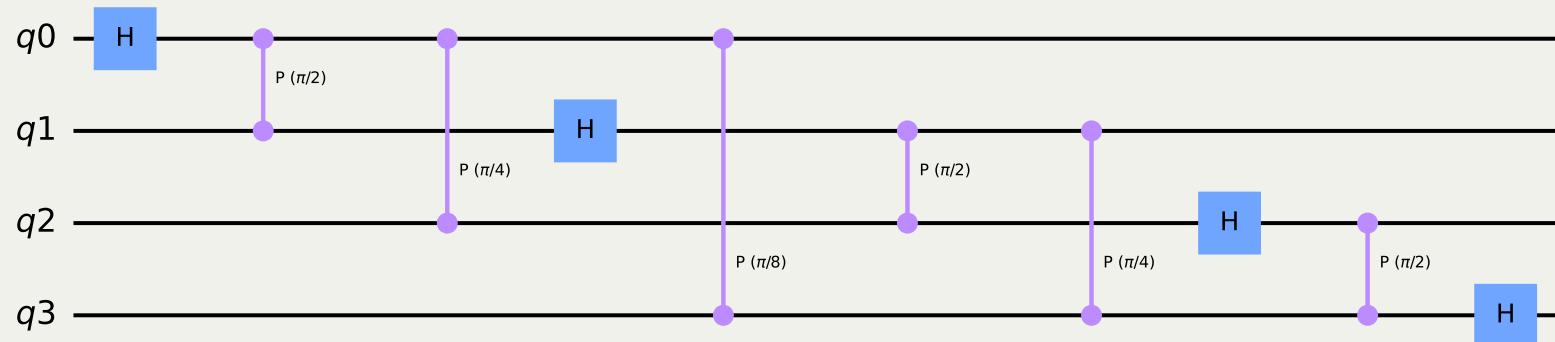
Quantum Fourier Transform

```
1 # QFT circuit
2 n = 4
3 qft = (
4     Qpivot(mapping=Qunitary("h()^0"))
5     + Qpivot(
6         mapping=Qunitary("cp(x)^01"),
7         share_weights=False,
8         symbol_fn=lambda x, ns, ne: np.pi * 2 ** (-ne),
9     )
10    + Qmask("1*")
11 ) * n
12 qft_n = Qinit(n) + qft
13
14 circuit_qft = qft_n(backend="qiskit", barriers=False)
15 circuit_qft.draw("mpl")
```



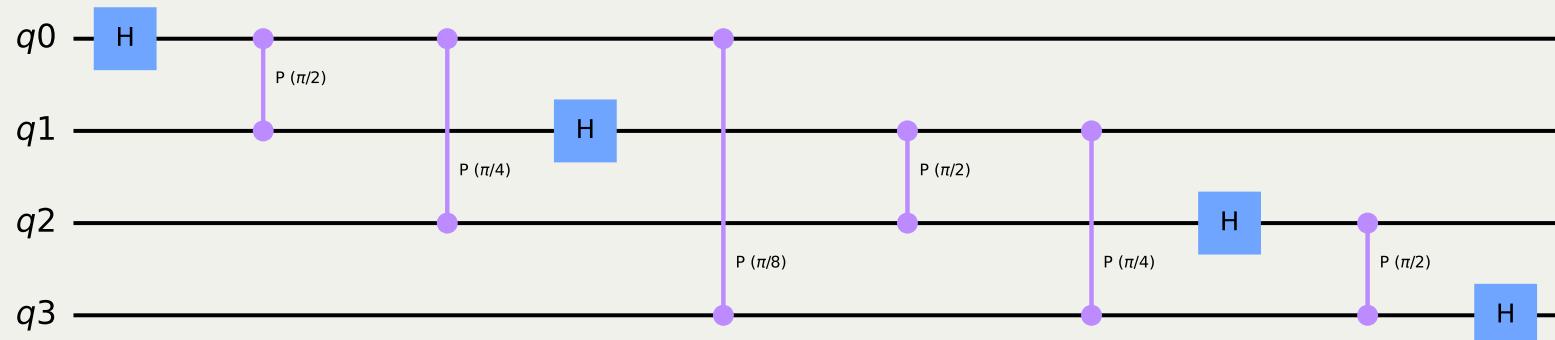
Quantum Fourier Transform

```
1 # QFT circuit
2 n = 4
3 qft = (
4     Qpivot(mapping=Qunitary("h()^0"))
5     + Qpivot(
6         mapping=Qunitary("cp(x)^01"),
7         share_weights=False,
8         symbol_fn=lambda x, ns, ne: np.pi * 2 ** (-ne),
9     )
10    + Qmask("1*")
11 ) * n
12 qft_n = Qinit(n) + qft
13
14 circuit_qft = qft_n(backend="qiskit", barriers=False)
15 circuit_qft.draw("mpl")
```



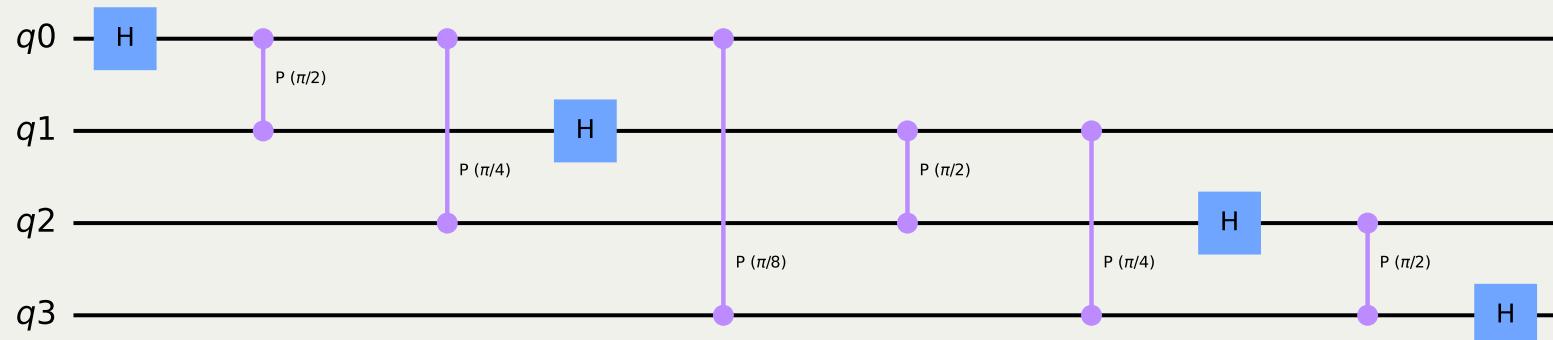
Quantum Fourier Transform

```
1 # QFT circuit
2 n = 4
3 qft = (
4     Qpivot(mapping=Qunitary("h()^0"))
5     + Qpivot(
6         mapping=Qunitary("cp(x)^01"),
7         share_weights=False,
8         symbol_fn=lambda x, ns, ne: np.pi * 2 ** (-ne),
9     )
10    + Qmask("1*")
11 ) * n
12 qft_n = Qinit(n) + qft
13
14 circuit_qft = qft_n(backend="qiskit", barriers=False)
15 circuit_qft.draw("mpl")
```



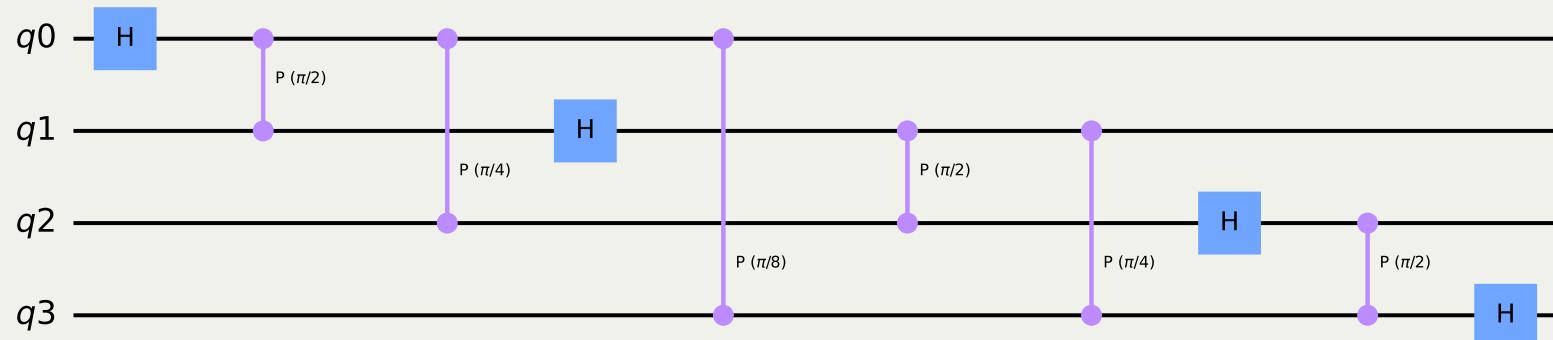
Quantum Fourier Transform

```
1 # QFT circuit
2 n = 4
3 qft = (
4     Qpivot(mapping=Qunitary("h()^0"))
5     + Qpivot(
6         mapping=Qunitary("cp(x)^01"),
7         share_weights=False,
8         symbol_fn=lambda x, ns, ne: np.pi * 2 ** (-ne),
9     )
10    + Qmask("1*")
11 ) * n
12 qft_n = Qinit(n) + qft
13
14 circuit_qft = qft_n(backend="qiskit", barriers=False)
15 circuit_qft.draw("mpl")
```



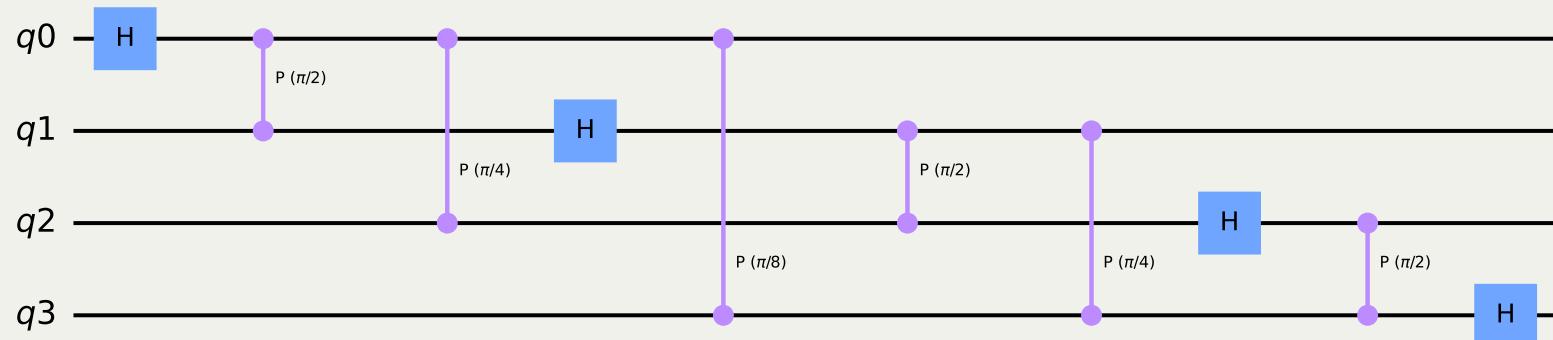
Quantum Fourier Transform

```
1 # QFT circuit
2 n = 4
3 qft = (
4     Qpivot(mapping=Qunitary("h()^0"))
5     + Qpivot(
6         mapping=Qunitary("cp(x)^01"),
7         share_weights=False,
8         symbol_fn=lambda x, ns, ne: np.pi * 2 ** (-ne),
9     )
10    + Qmask("1*")
11 ) * n
12 qft_n = Qinit(n) + qft
13
14 circuit_qft = qft_n(backend="qiskit", barriers=False)
15 circuit_qft.draw("mpl")
```



Quantum Fourier Transform

```
1 # QFT circuit
2 n = 4
3 qft = (
4     Qpivot(mapping=Qunitary("h()^0"))
5     + Qpivot(
6         mapping=Qunitary("cp(x)^01"),
7         share_weights=False,
8         symbol_fn=lambda x, ns, ne: np.pi * 2 ** (-ne),
9     )
10    + Qmask("1*")
11 ) * n
12 qft_n = Qinit(n) + qft
13
14 circuit_qft = qft_n(backend="qiskit", barriers=False)
15 circuit_qft.draw("mpl")
```



Classical Adder

Classical Adder

```
1 def half_adder(bits, symbols=None, state=None):
2     b1, b2 = state[bits[0]], state[bits[1]]
3     xor = b1 ^ b2
4     carry = b1 and b2
5     state[bits[0]] = carry
6     state[bits[1]] = xor
7     return state
8
9 def or_top(bits, symbols=None, state=None):
10    b1, b2 = state[bits[0]], state[bits[1]]
11    state[bits[0]] = b1 or b2
12    return state
```

<https://github.com/matt-lourens/hierarqcal>

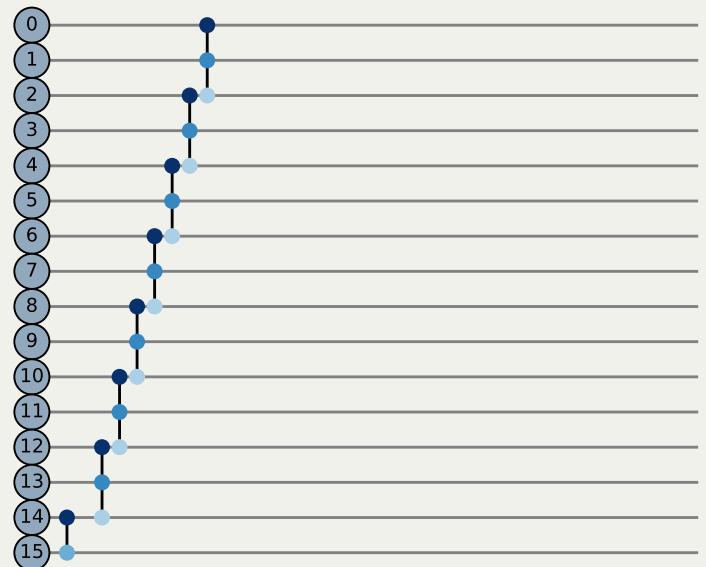
Classical Adder

```
1 def half_adder(bits, symbols=None, state=None):
2     b1, b2 = state[bits[0]], state[bits[1]]
3     xor = b1 ^ b2
4     carry = b1 and b2
5     state[bits[0]] = carry
6     state[bits[1]] = xor
7     return state
8
9 def or_top(bits, symbols=None, state=None):
10    b1, b2 = state[bits[0]], state[bits[1]]
11    state[bits[0]] = b1 or b2
12    return state
```

<https://github.com/matt-lourens/hierarqcal>

Classical Adder

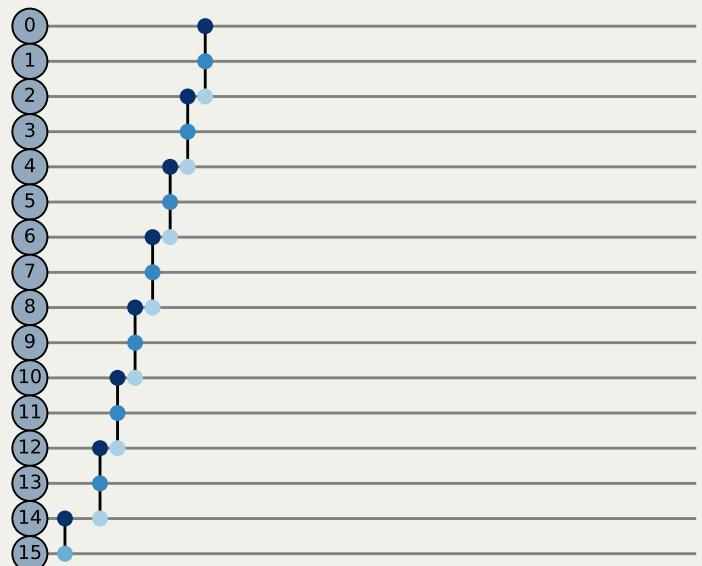
```
1 # program
2 full_adder = (
3     Qinit(3)
4     + Qcycle(mapping=half_adder, boundary="open")
5     + Qpivot(global_pattern="1*", merge_within="11"
6             ,mapping=or_top)
7 )
8 addition = (
9     Qinit(n)
10    + Qpivot("*1", "11", mapping=half_adder)
11    + Qcycle(step=2,
12              edge_order=[-1],
13              mapping=full_adder,
14              boundary="open")
15 )
```



<https://github.com/matt-lourens/hierarqcal>

Classical Adder

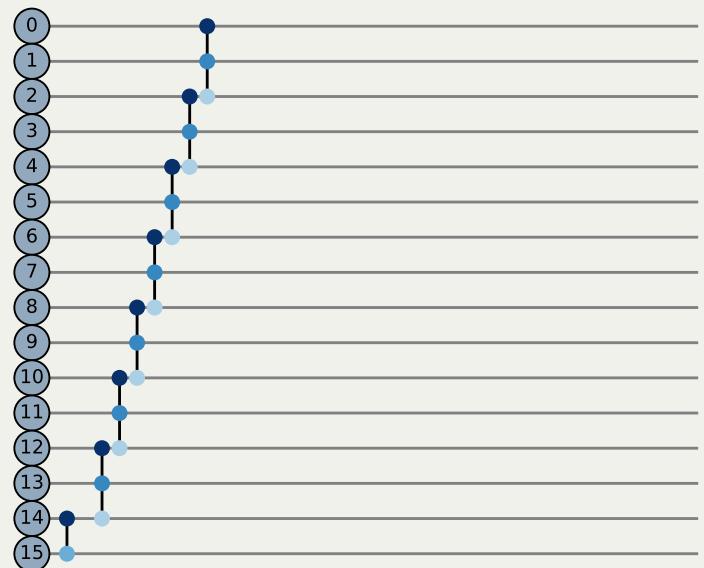
```
1 # program
2 full_adder = (
3     Qinit(3)
4     + Qcycle(mapping=half_adder, boundary="open")
5     + Qpivot(global_pattern="1*", merge_within="11"
6             ,mapping=or_top)
7 )
8 addition = (
9     Qinit(n)
10    + Qpivot("*1", "11", mapping=half_adder)
11    + Qcycle(step=2,
12              edge_order=[-1],
13              mapping=full_adder,
14              boundary="open")
15 )
```



<https://github.com/matt-lourens/hierarqcal>

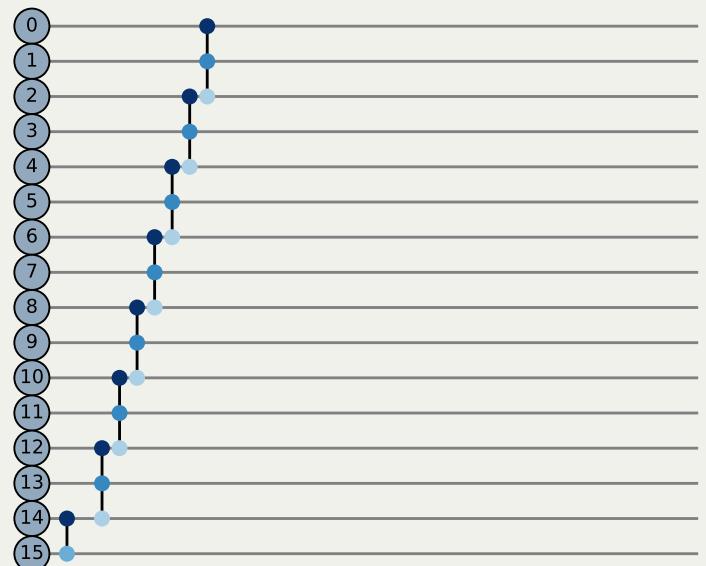
Classical Adder

```
1 # program
2 full_adder = (
3     Qinit(3)
4     + Qcycle(mapping=half_adder, boundary="open")
5     + Qpivot(global_pattern="1*", merge_within="11"
6             ,mapping=or_top)
7 )
8 addition = (
9     Qinit(n)
10    + Qpivot("*1", "11", mapping=half_adder)
11    + Qcycle(step=2,
12              edge_order=[-1],
13              mapping=full_adder,
14              boundary="open")
15 )
```



Classical Adder

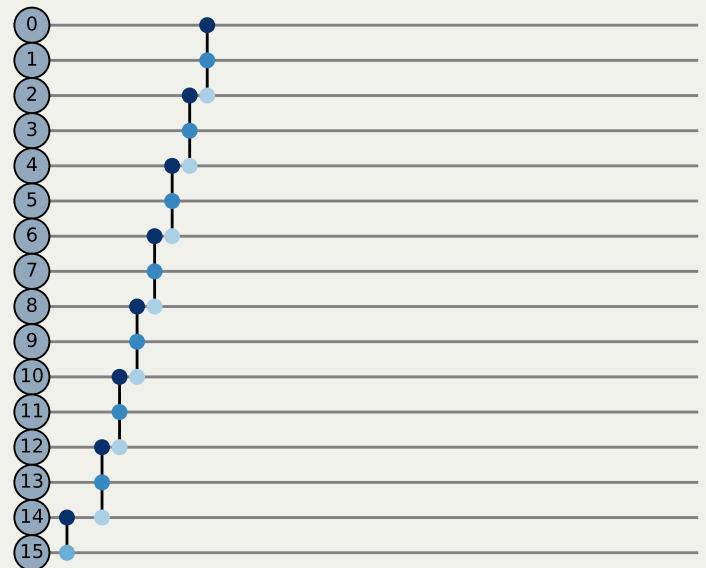
```
1 # program
2 full_adder = (
3     Qinit(3)
4     + Qcycle(mapping=half_adder, boundary="open")
5     + Qpivot(global_pattern="1*", merge_within="11"
6             ,mapping=or_top)
7 )
8 addition = (
9     Qinit(n)
10    + Qpivot("*1", "11", mapping=half_adder)
11    + Qcycle(step=2,
12              edge_order=[-1],
13              mapping=full_adder,
14              boundary="open")
15 )
```



<https://github.com/matt-lourens/hierarqcal>

Classical Adder

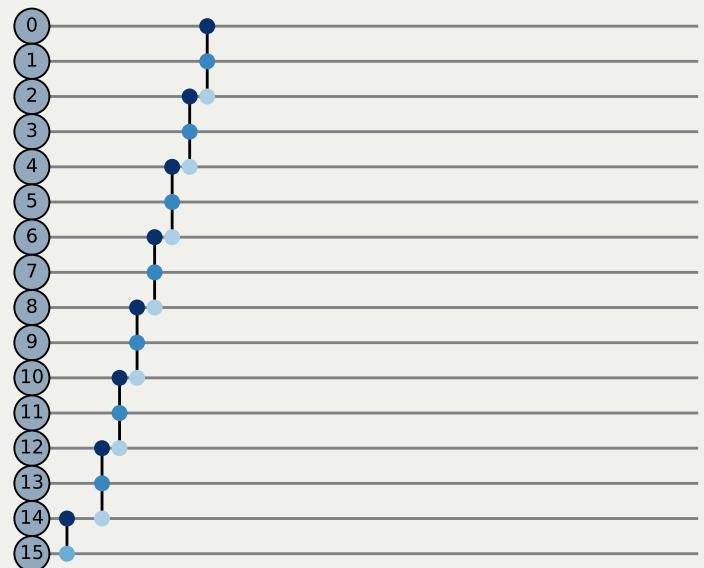
```
1 # program
2 full_adder = (
3     Qinit(3)
4     + Qcycle(mapping=half_adder, boundary="open")
5     + Qpivot(global_pattern="1*", merge_within="11"
6             ,mapping=or_top)
7 )
8 addition = (
9     Qinit(n)
10    + Qpivot("*1", "11", mapping=half_adder)
11    + Qcycle(step=2,
12              edge_order=[-1],
13              mapping=full_adder,
14              boundary="open")
15 )
```



<https://github.com/matt-lourens/hierarqcal>

Classical Adder

```
1 # program
2 full_adder = (
3     Qinit(3)
4     + Qcycle(mapping=half_adder, boundary="open")
5     + Qpivot(global_pattern="1*", merge_within="11"
6             ,mapping=or_top)
7 )
8 addition = (
9     Qinit(n)
10    + Qpivot("*1", "11", mapping=half_adder)
11    + Qcycle(step=2,
12              edge_order=[-1],
13              mapping=full_adder,
14              boundary="open")
15 )
```



<https://github.com/matt-lourens/hierarqcal>

Paper



Package

