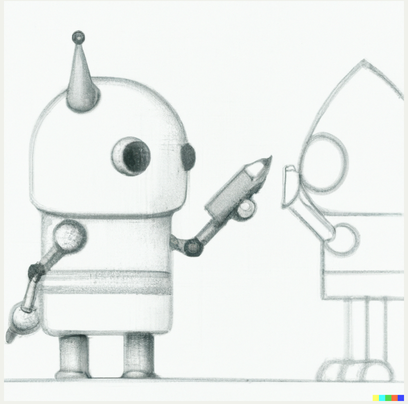


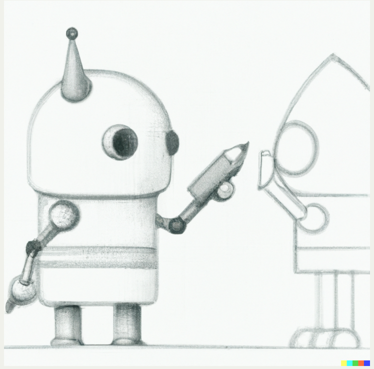
A cute robot building itself with artificial intelligence, pencil drawing - DALL·E 2

Hierarchical Quantum Circuit Representations

Stellenbosch University Iyunivesithi Universiteit

Oppenheimer Memorial Trust

Unitary Fund



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

Hierarchical Quantum Circuit Representations

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

Stellenbosch UNIVERSITY IYUNIVESITHI UNIVERSITEIT

OPPENHEIMER MEMORIAL TRUST

Unitary Fund

Outline

HierarQcal

Design and implement hierarchical compute graphs

Hierarchical Quantum Circuit Representations

Theoretical framework behind package

This talk

- Background
- Overview of Representation
- Results and usage

Background

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.

Krizhevsky, A., Sutskever, I. & Hinton, G. E. Imagenet classification with deep convolutional neural networks. Adv. Neural Inf. Process. Syst. 25 (2012).

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
Sparse coding [2]	*47.1%*	*28.2%*
SIFT + FVs [24]	*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.

Krizhevsky, A., Sutskever, I. & Hinton, G. E. Imagenet classification with deep convolutional neural networks. Adv. Neural Inf. Process. Syst. 25 (2012).

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
Sparse coding [2]	*47.1%*	*28.2%*
SIFT + FVs [24]	*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.

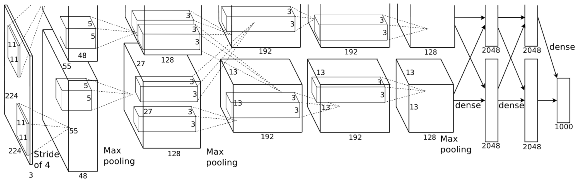
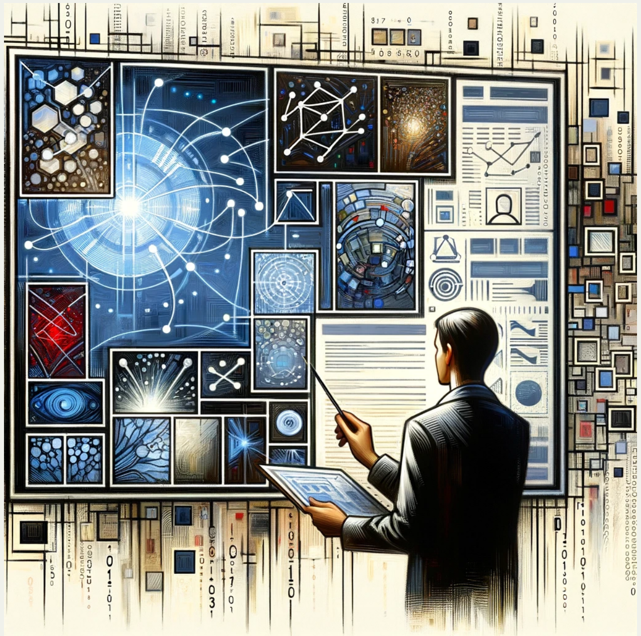


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input 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.

Krizhevsky, A., Sutskever, I. & Hinton, G. E. Imagenet classification with deep convolutional neural networks. Adv. Neural Inf. Process. Syst. 25 (2012).

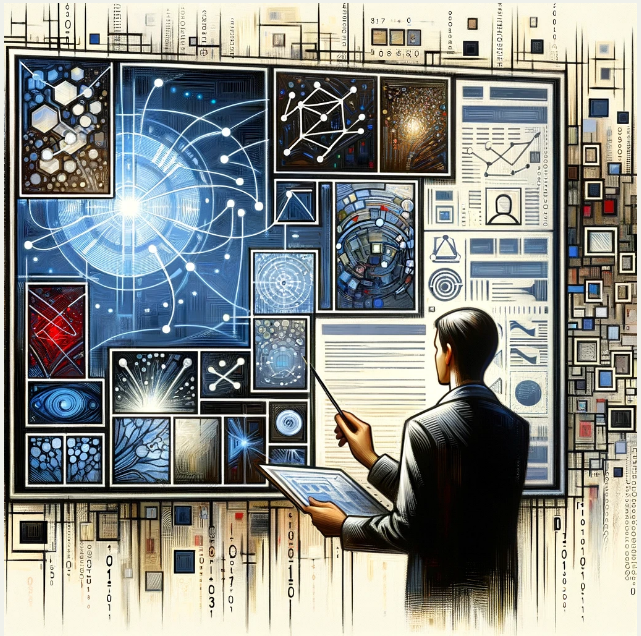
Automation



Images Generated with DALL·E 3

Automation

Manual



→

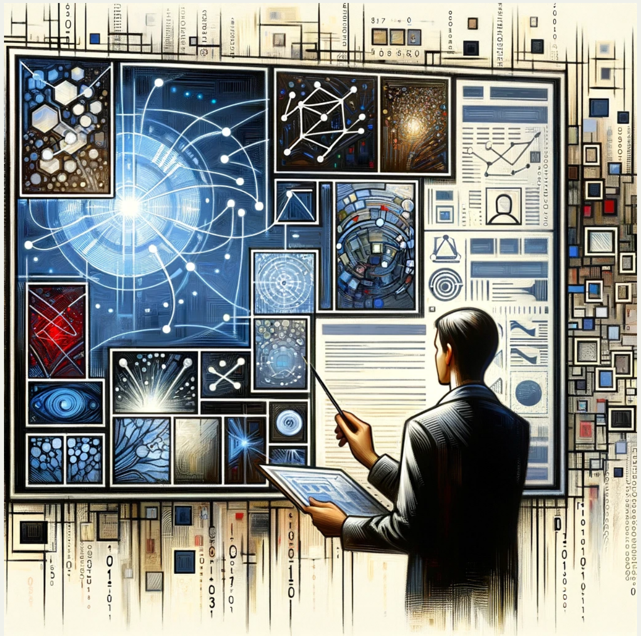
Feature Engineering

Automated

Images Generated with DALL·E 3

Automation

Manual



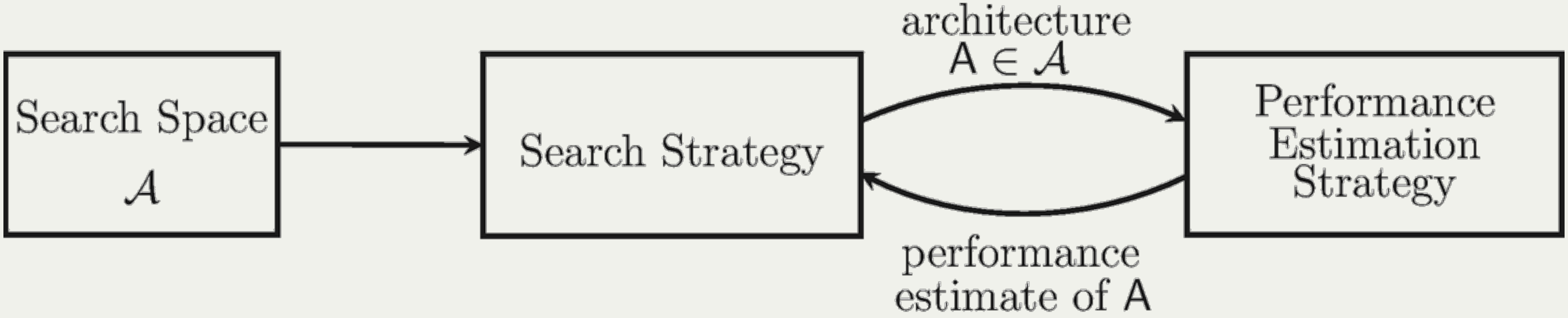
→

Architecture Engineering?

Automated

Images Generated with DALL·E 3

Neural Architecture Search



*[*] **Fig:** Abstract illustration of Neural Architecture Search methods. A search strategy selects an architecture A from a predefined search space 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

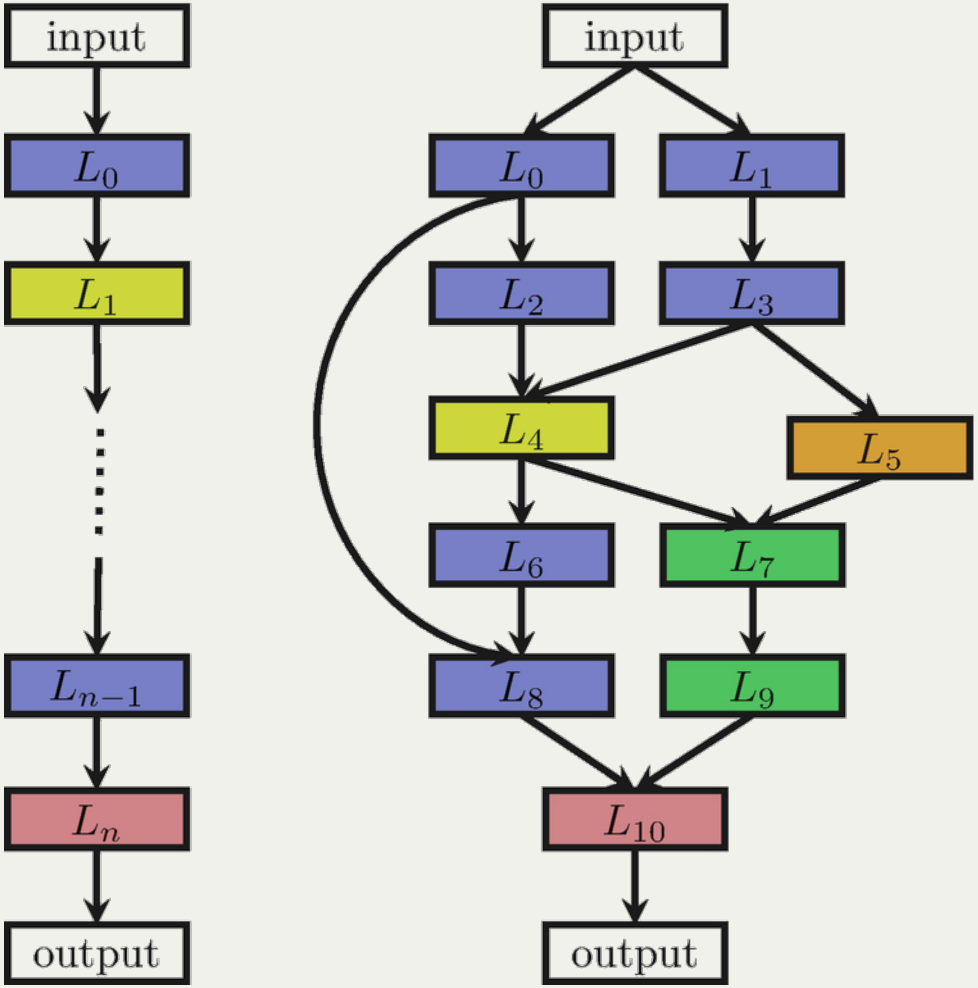


***Fig:** An illustration of different architecture spaces. Each node in the graphs corresponds to a layer in a neural network, e.g., a convolutional or pooling layer. Different layer types are visualized by different colors. An edge from layer L_i to layer L_j denotes that L_j receives the output of 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

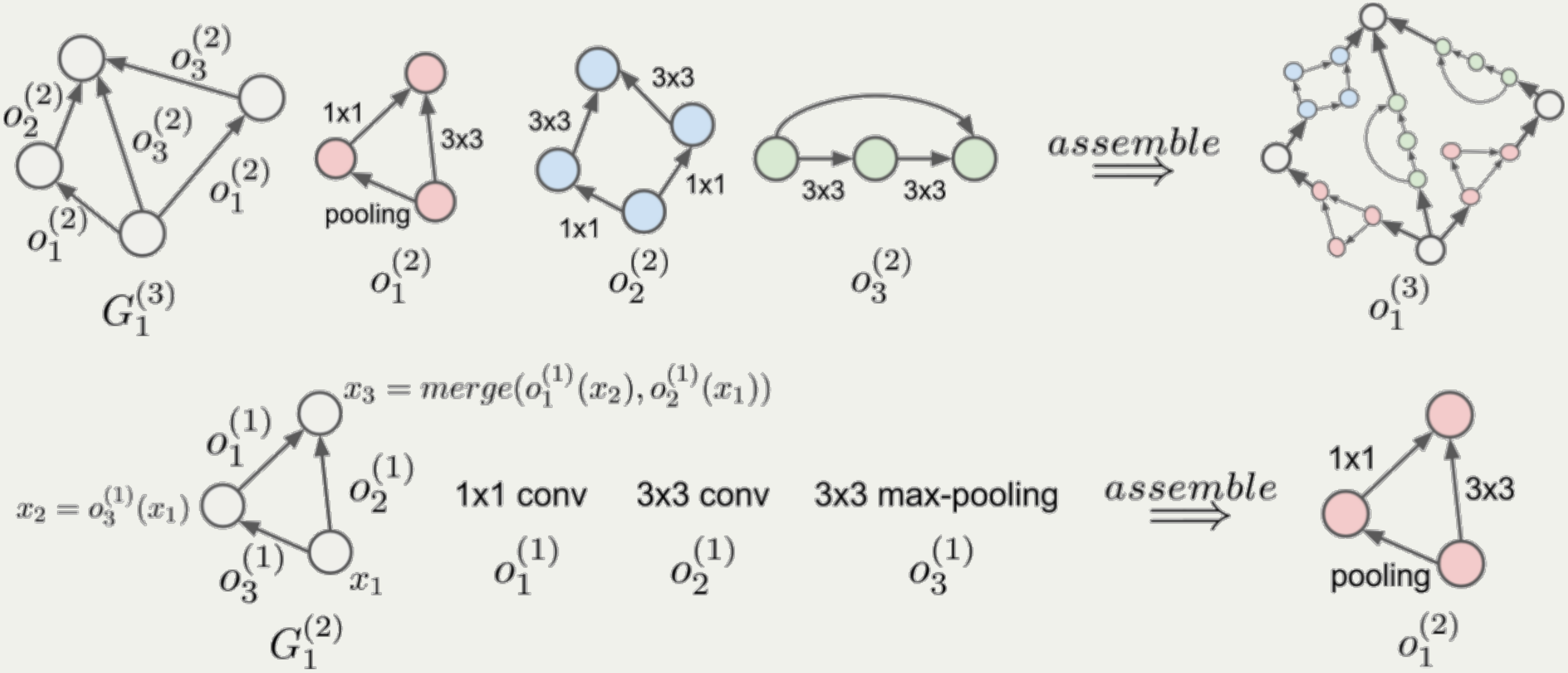


*Fig:** An example of a three-level hierarchical architecture representation. The bottom row shows how level-1 primitive operations $o_1^{(1)}$, $o_2^{(1)}$, $o_3^{(1)}$ are assembled into a level-2 motif $o_1^{(2)}$. The top row shows how level-2 motifs $o_1^{(2)}$, $o_2^{(2)}$, $o_3^{(2)}$ are then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

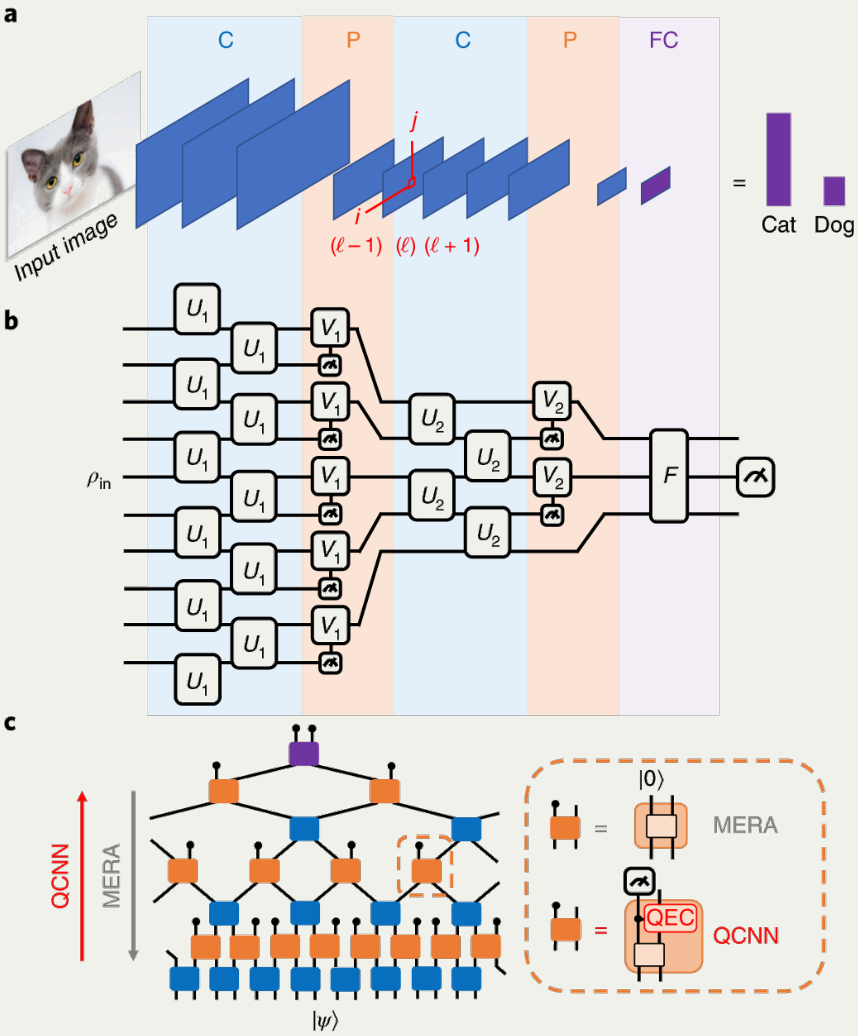


*Fig: The concept of QCNNs. *a*, Simplified illustration of classical CNNs. A sequence of image-processing layers transforms an input image into a series of feature maps (blue rectangles) and finally into an output probability distribution (purple bars). C, convolution; P, pooling; FC, fully connected. *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/.

Ansatz Design

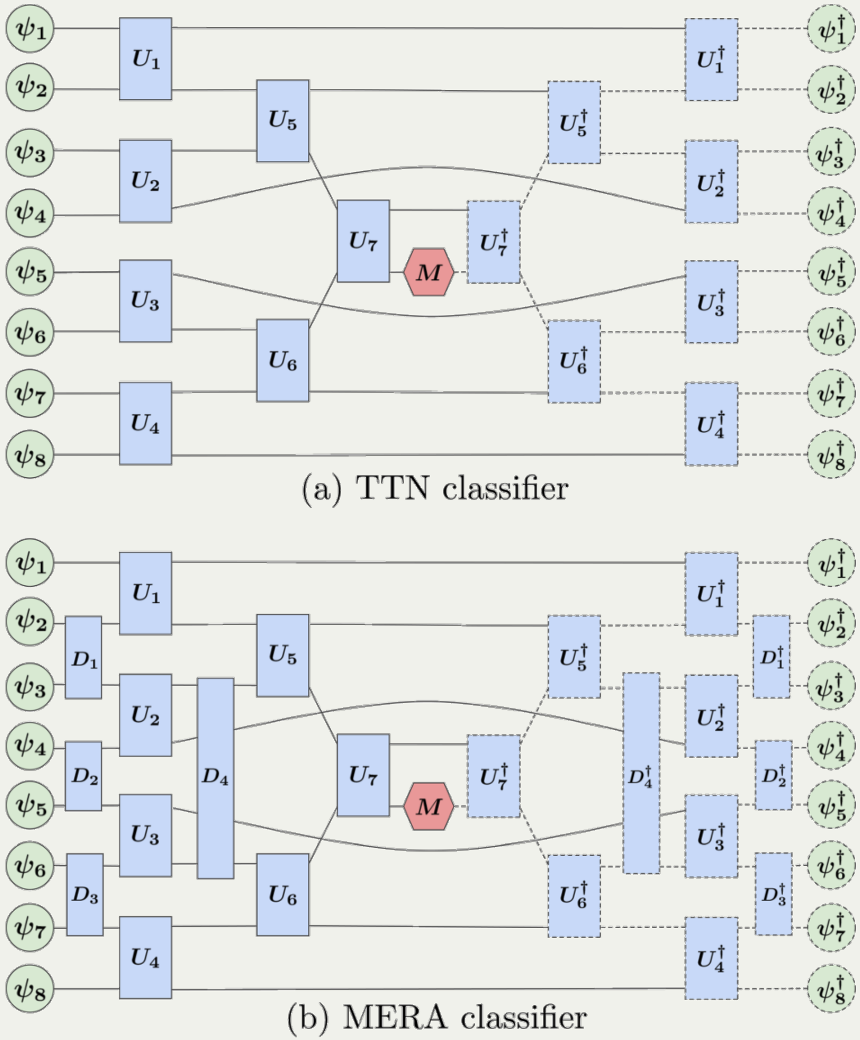
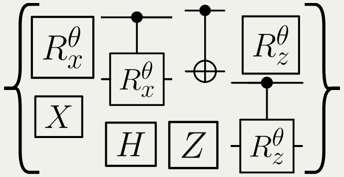


*****Fig:** TTN and MERA classifiers for eight qubits. The quantum circuit is illustrated by the regions outlined in solid lines comprising inputs ψ, unitary blocks $\{U_i\}_{i=1}^{7}$ and $\{D_i\}_{i=1}^{4}$, and a measurement operator M. The dashed lines represent its conjugate transpose. The solid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

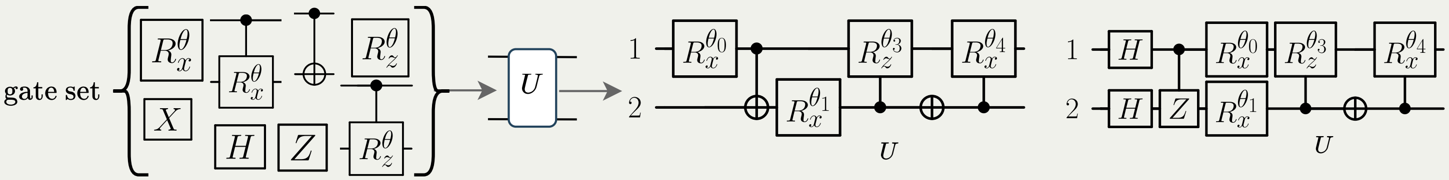
*Grant, E. et al. Hierarchical quantum classifiers. NPJ Quantum Inf. 4, 65 (2018).

Representation

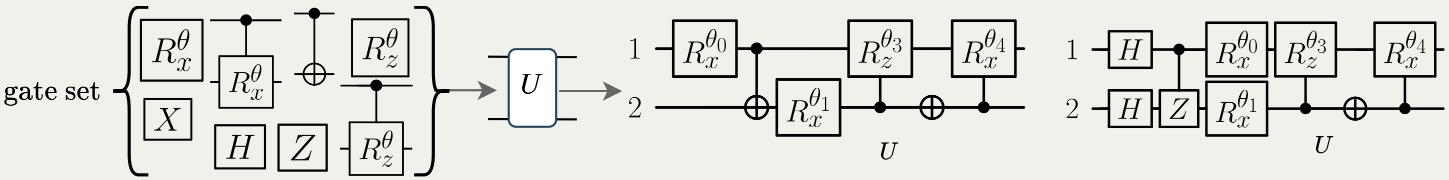
Motifs

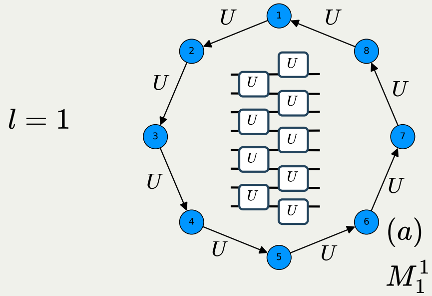
gate set 

Motifs



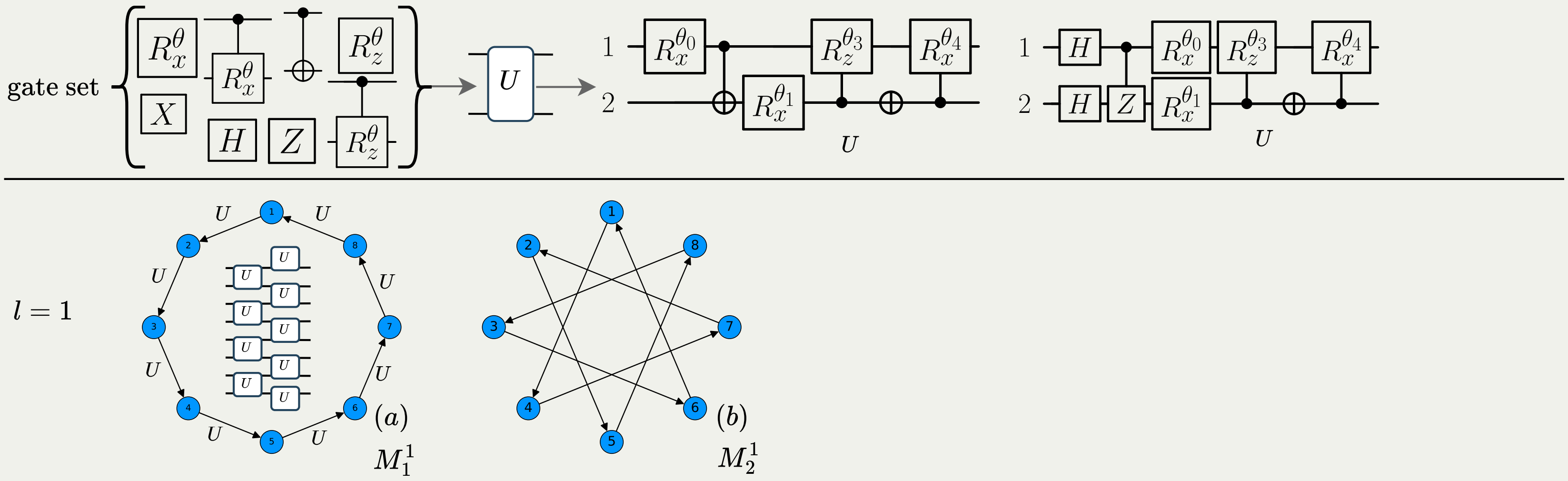
Motifs





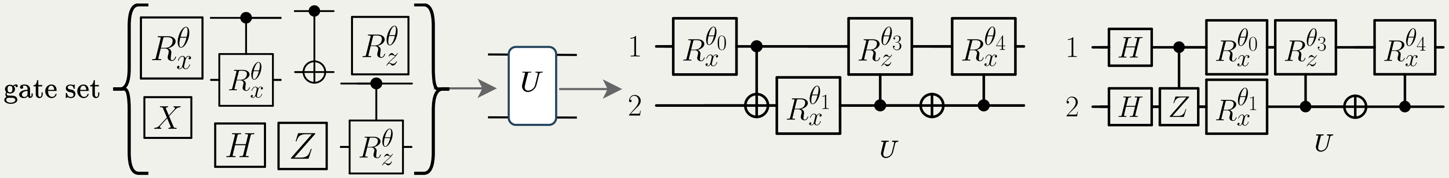

```
(a): Qcycle(stride=1)
```

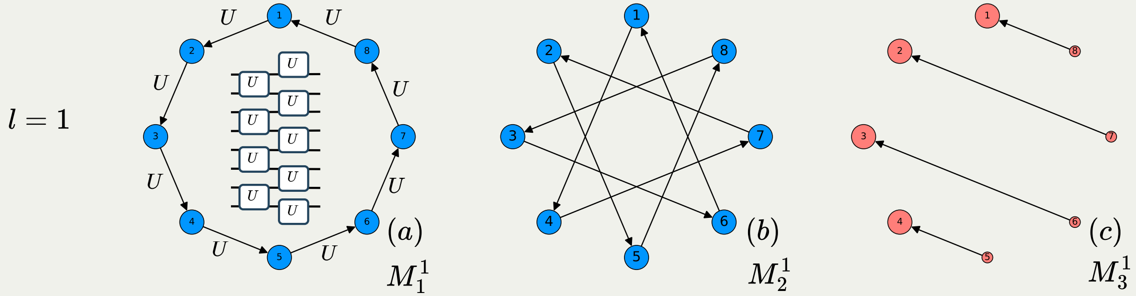
Motifs




```
(a): Qcycle(stride=1)
(b): Qcycle(stride=3)
```

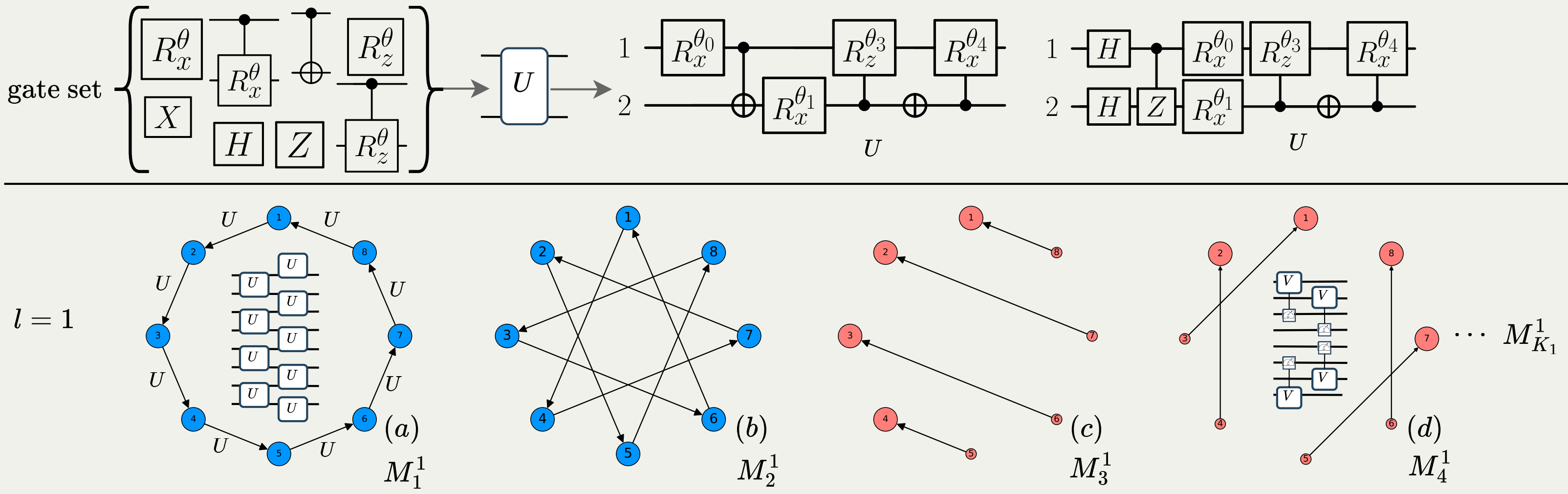
Motifs





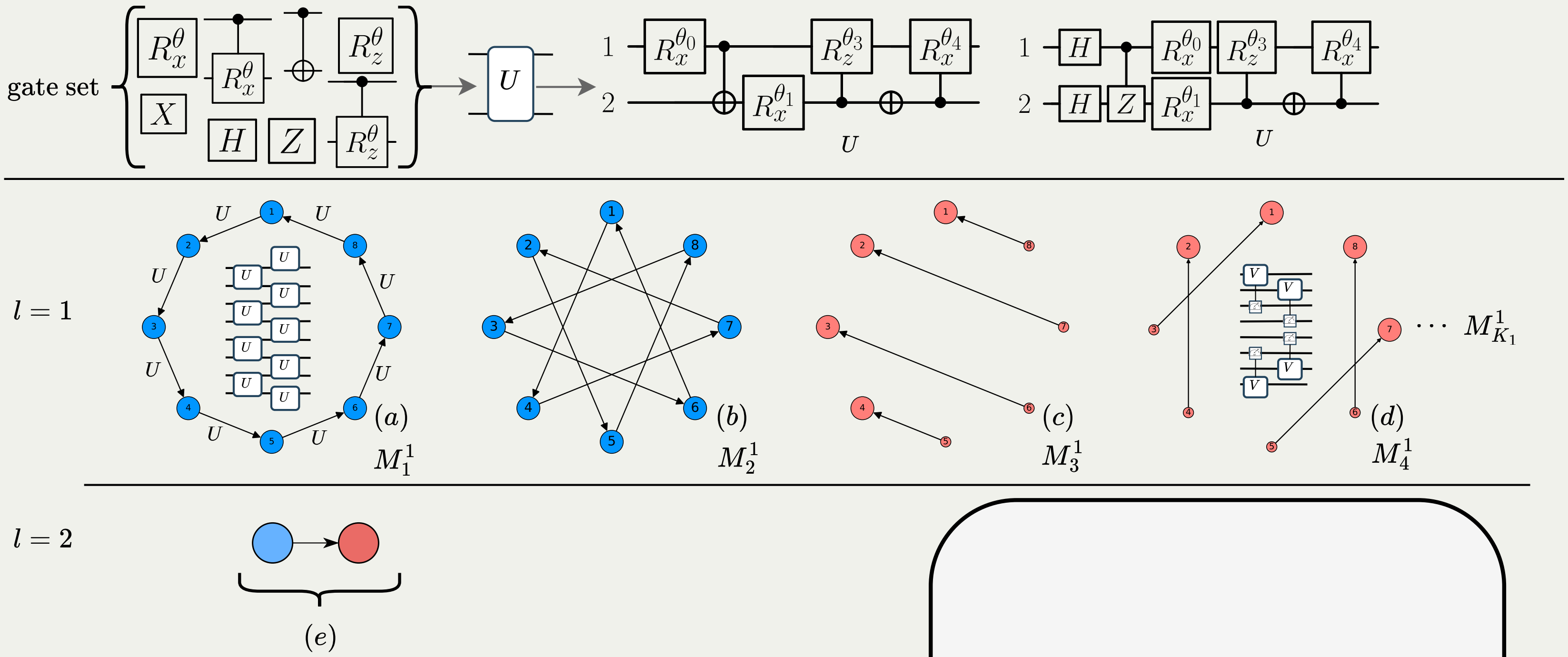

```
(a): Qcycle(stride=1)
(b): Qcycle(stride=3)
(c): Qmask(global_pattern="*!")
```

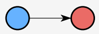
Motifs




```
(a): Qcycle(stride=1)
(b): Qcycle(stride=3)
(c): Qmask(global_pattern="*!")
(d): Qmask(global_pattern="*!*")
```

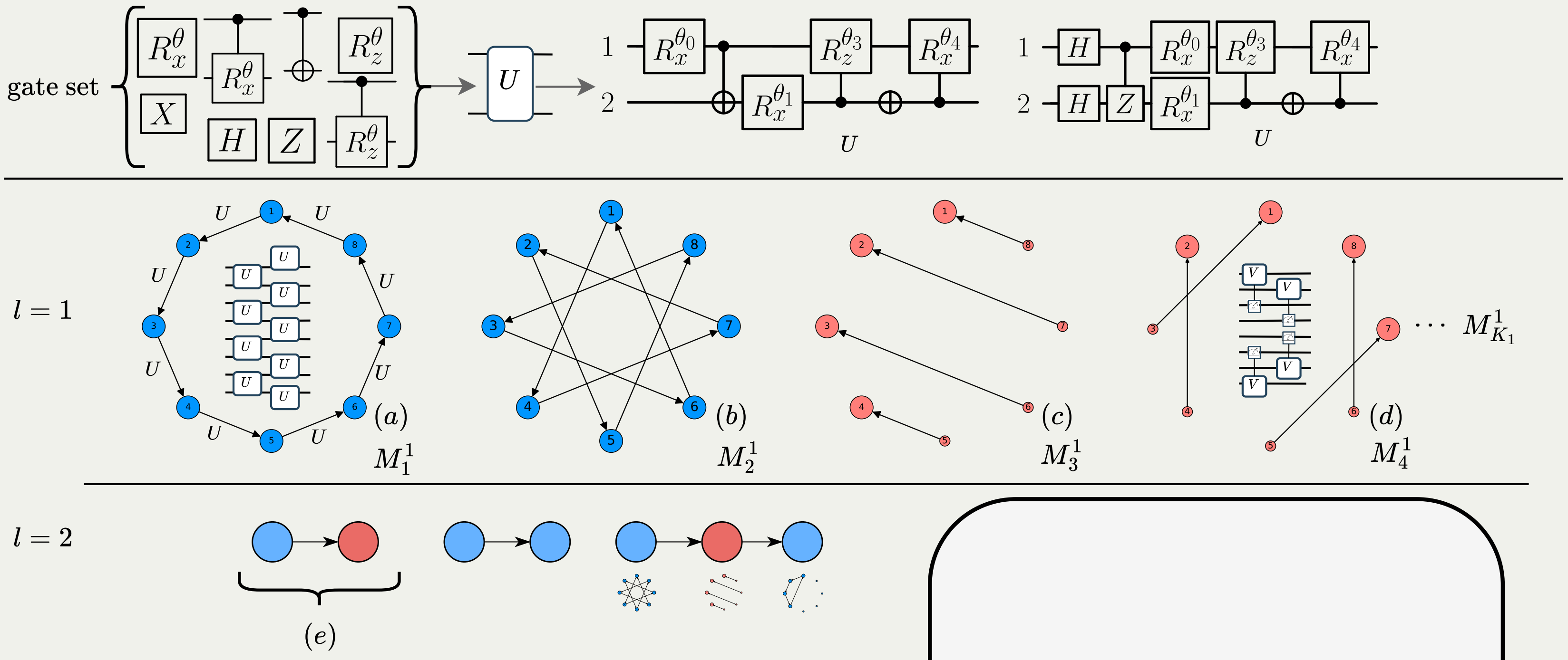
Motifs

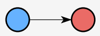




```
(a): Qcycle(stride=1)
(b): Qcycle(stride=3)
(c): Qmask(global_pattern="*!")
(d): Qmask(global_pattern="*!*")
(e): m1 = Qcycle(1) + Qmask("*!")
```

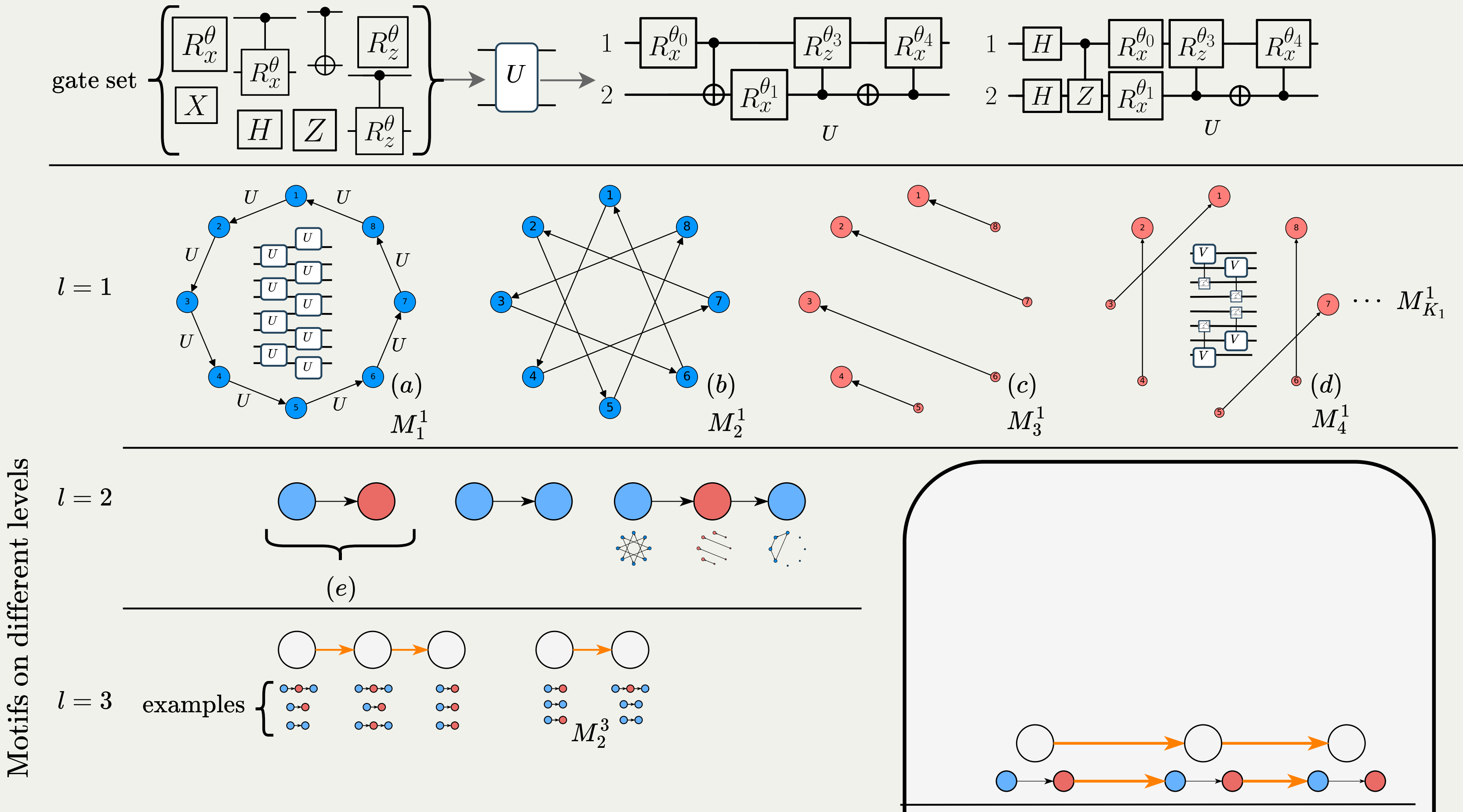
Motifs





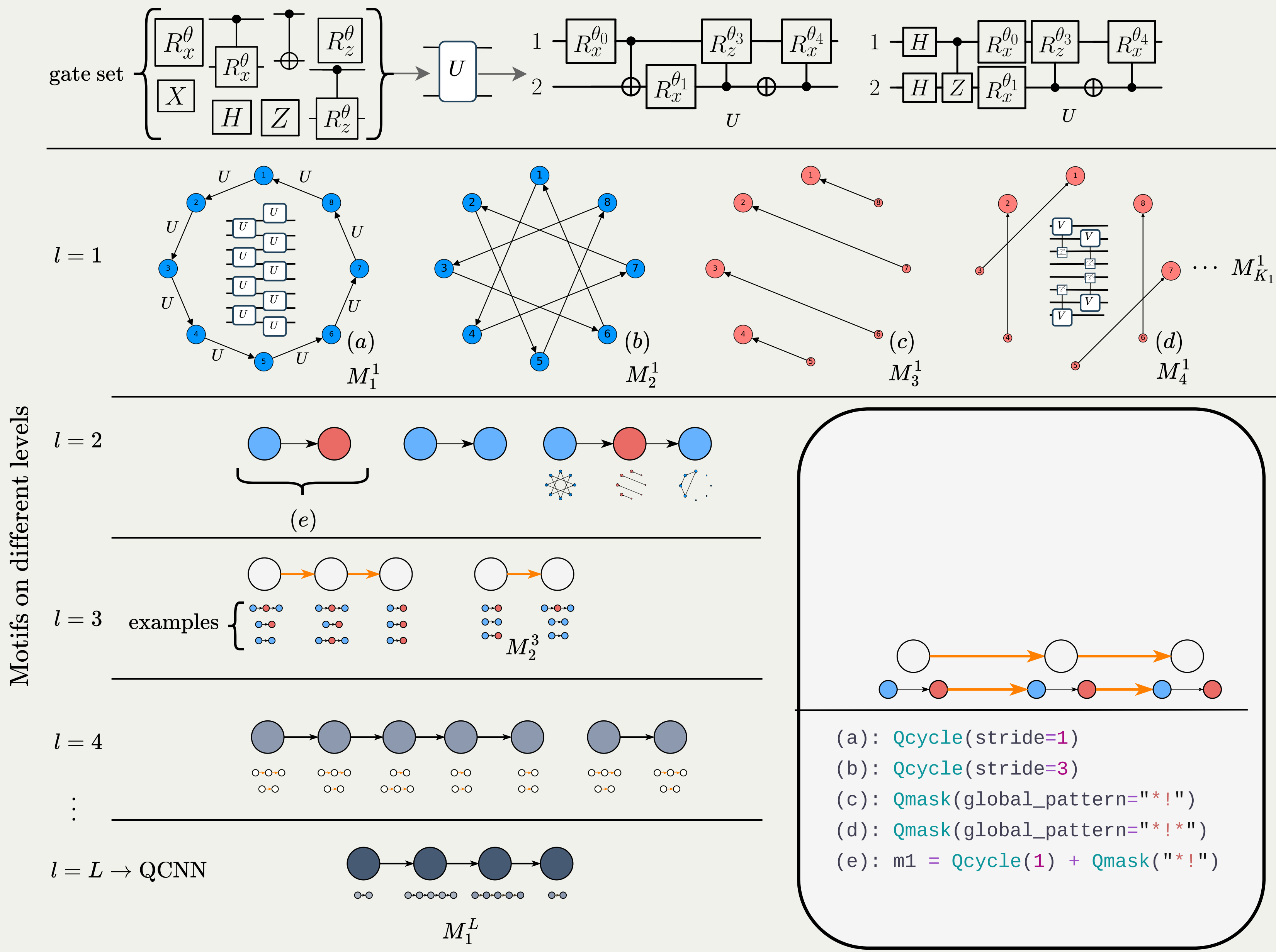
```
(a): Qcycle(stride=1)
(b): Qcycle(stride=3)
(c): Qmask(global_pattern="*!")
(d): Qmask(global_pattern="*!*")
(e): m1 = Qcycle(1) + Qmask("*!")
```

Motifs



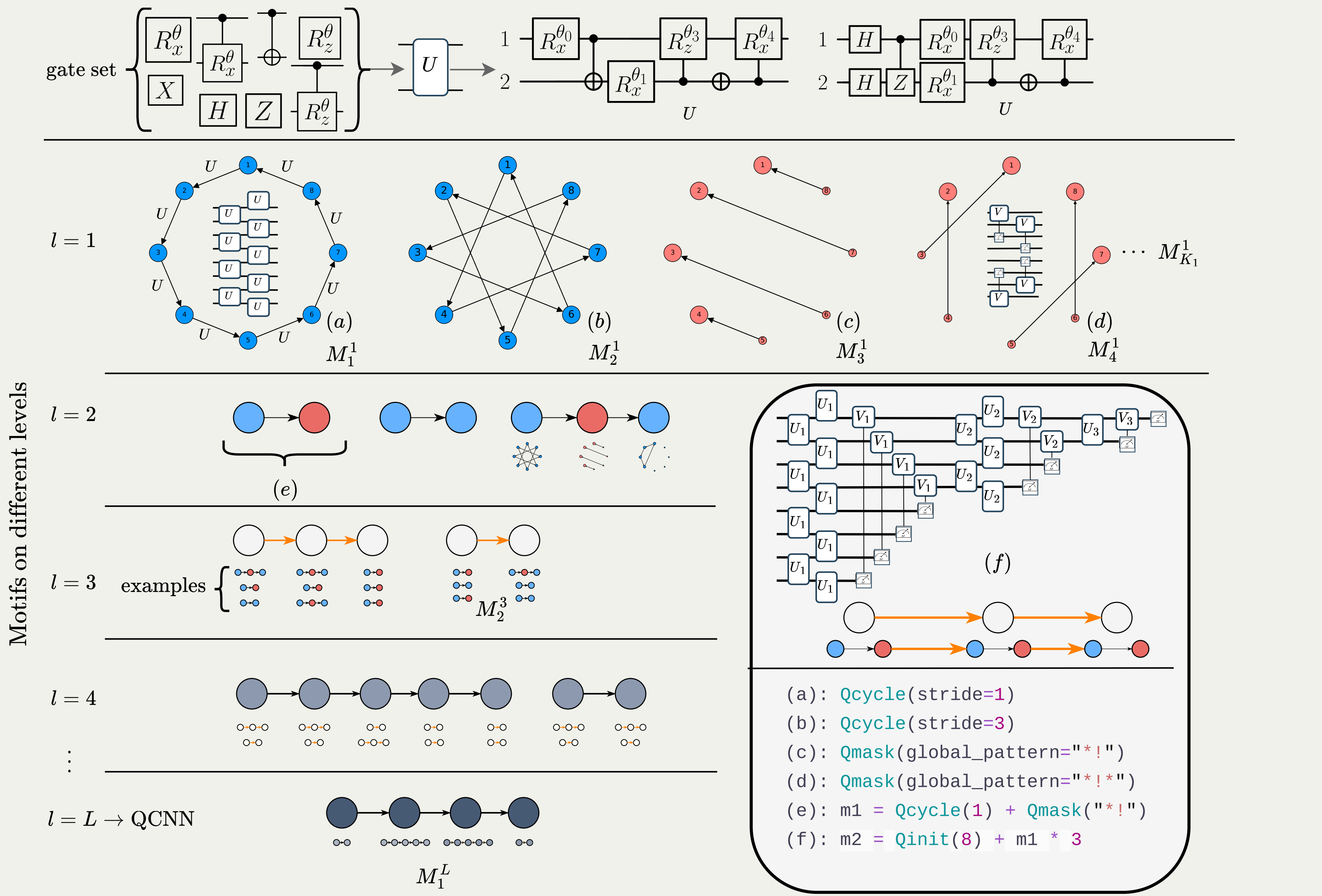

```
(a): Qcycle(stride=1)
(b): Qcycle(stride=3)
(c): Qmask(global_pattern="*!")
(d): Qmask(global_pattern="*!*")
(e): m1 = Qcycle(1) + Qmask("*!")
```

Motifs



gate set
U
l = 1
(a)
(b)
(c)
(d)
M_1^1
M_2^1
M_3^1
M_4^1
$\cdots M_{K_1}^1$
l = 2
(e)
l = 3
examples
M_2^3
l = 4
l = L → QCNN
M_1^L
Motifs on different levels
(a): Qcycle(stride=1)
(b): Qcycle(stride=3)
(c): Qmask(global_pattern="*!")
(d): Qmask(global_pattern="*!*")
(e): m1 = Qcycle(1) + Qmask("*!")

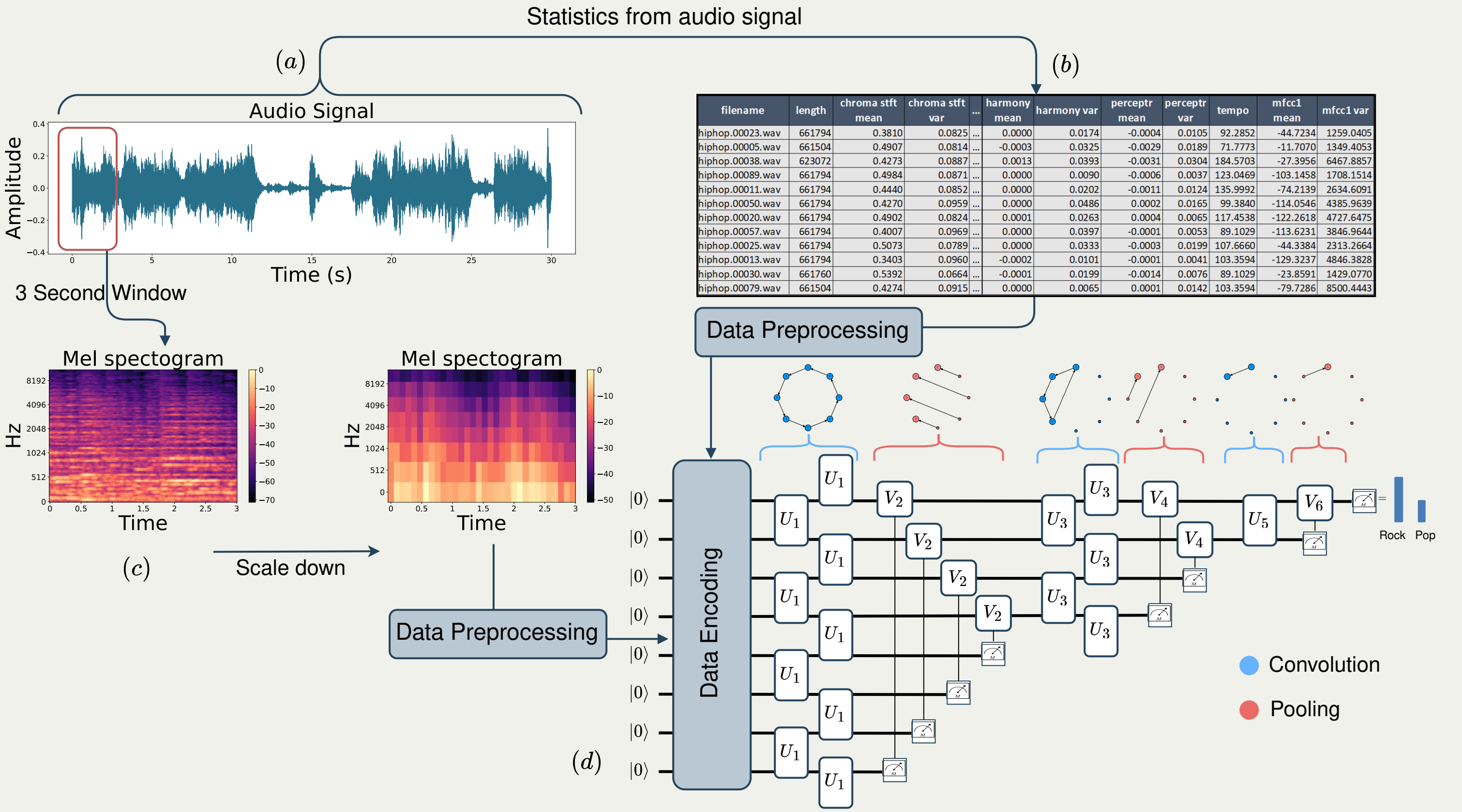
Motifs



gate set
U
Motifs on different levels
$l = 1$
(a) M_1^1
(b) M_2^1
(c) M_3^1
(d) M_4^1
$\cdots M_{K_1}^1$
$l = 2$
(e)
$l = 3$
examples
M_2^3
$l = 4$
$l = L \rightarrow$ QCNN
M_1^L
(f)
(a): Qcycle(stride=1)
(b): Qcycle(stride=3)
(c): Qmask(global_pattern="*!")
(d): Qmask(global_pattern="*!*")
(e): m1 = Qcycle(1) + Qmask("*!")
(f): m2 = Qinit(8) + m1 * 3

Results and Usage

Music Genre Classification

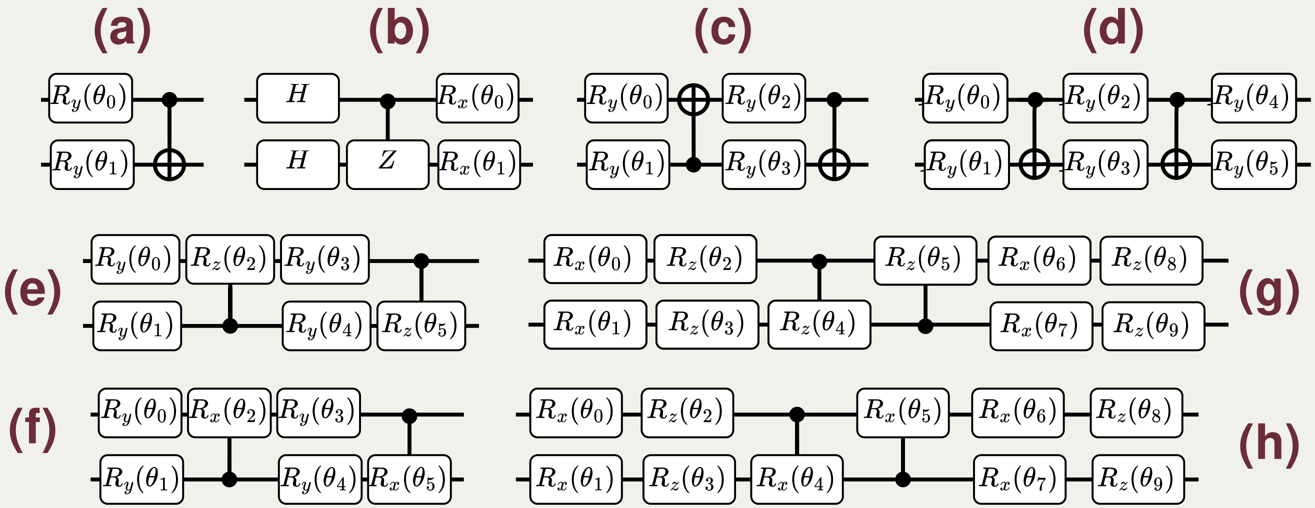


filename	length	chroma stft mean	chroma stft var	...	harmony mean	harmony var	perceptr mean	perceptr var	tempo	mfcc1 mean	mfcc1 var
hiphop.00023.wav	661794	0.3810	0.0825	...	0.0000	0.0174	-0.0004	0.0105	92.2852	-44.7234	1259.0405
hiphop.00005.wav	661504	0.4907	0.0814	...	-0.0003	0.0325	-0.0029	0.0189	71.7773	-11.7070	1349.4053
hiphop.00038.wav	623072	0.4273	0.0887	...	0.0013	0.0393	-0.0031	0.0304	184.5703	-27.3956	6467.8857
hiphop.00089.wav	661794	0.4984	0.0871	...	0.0000	0.0090	-0.0006	0.0037	123.0469	-103.1458	1708.1514
hiphop.00011.wav	661794	0.4440	0.0852	...	0.0000	0.0202	-0.0011	0.0124	135.9992	-74.2139	2634.6091
hiphop.00050.wav	661794	0.4270	0.0959	...	0.0000	0.0486	0.0002	0.0165	99.3840	-114.0546	4385.9639
hiphop.00020.wav	661794	0.4902	0.0824	...	0.0001	0.0263	0.0004	0.0065	117.4538	-122.2618	4727.6475
hiphop.00057.wav	661794	0.4007	0.0969	...	0.0000	0.0397	-0.0001	0.0053	89.1029	-113.6231	3846.9644
hiphop.00025.wav	661794	0.5073	0.0789	...	0.0000	0.0333	-0.0003	0.0199	107.6660	-44.3384	2313.2664
hiphop.00013.wav	661794	0.3403	0.0960	...	-0.0002	0.0101	-0.0001	0.0041	103.3594	-129.3237	4846.3828
hiphop.00030.wav	661760	0.5392	0.0664	...	-0.0001	0.0199	-0.0014	0.0076	89.1029	-23.8591	1429.0770
hiphop.00079.wav	661504	0.4274	0.0915	...	0.0000	0.0065	0.0001	0.0142	103.3594	-79.7286	8500.4443

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

Random Search

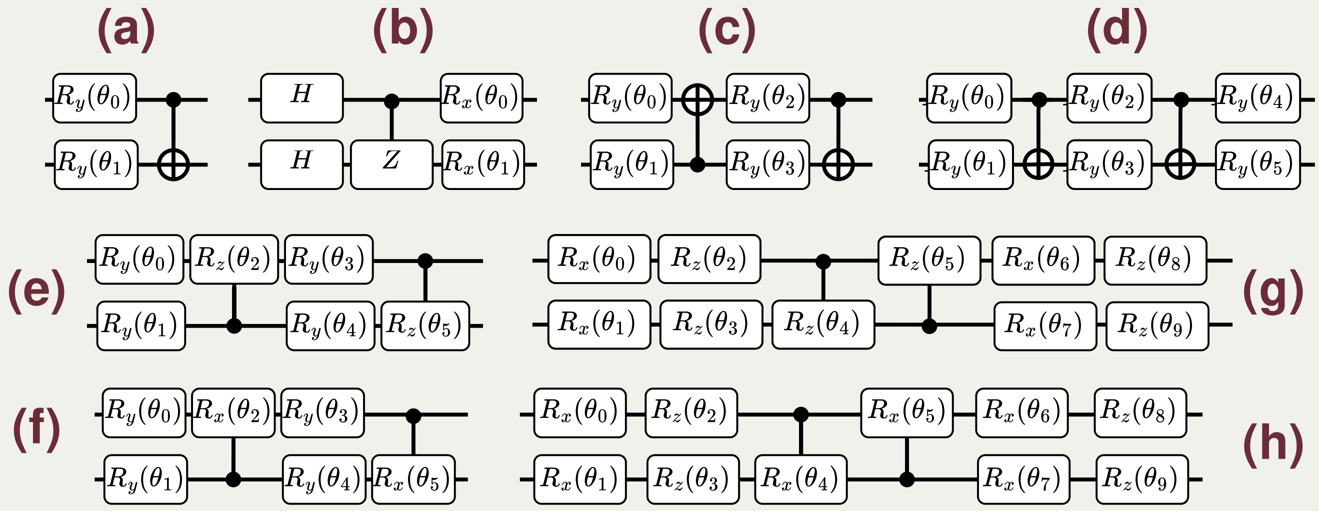
```python
# Example Search Space
strides = range(1, 8, 1)
steps = range(1, 8, 1)
mappings = [a, b, c, d, e, f, g, h]
boundaries = ["open", "periodic"]
patterns = ["10", "01", "*!", "!*", "!*!", "*!*"]

qcnn = Qinit(8) + (Qcycle(...) + Qmask(...)) * 3
```



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

Random Search

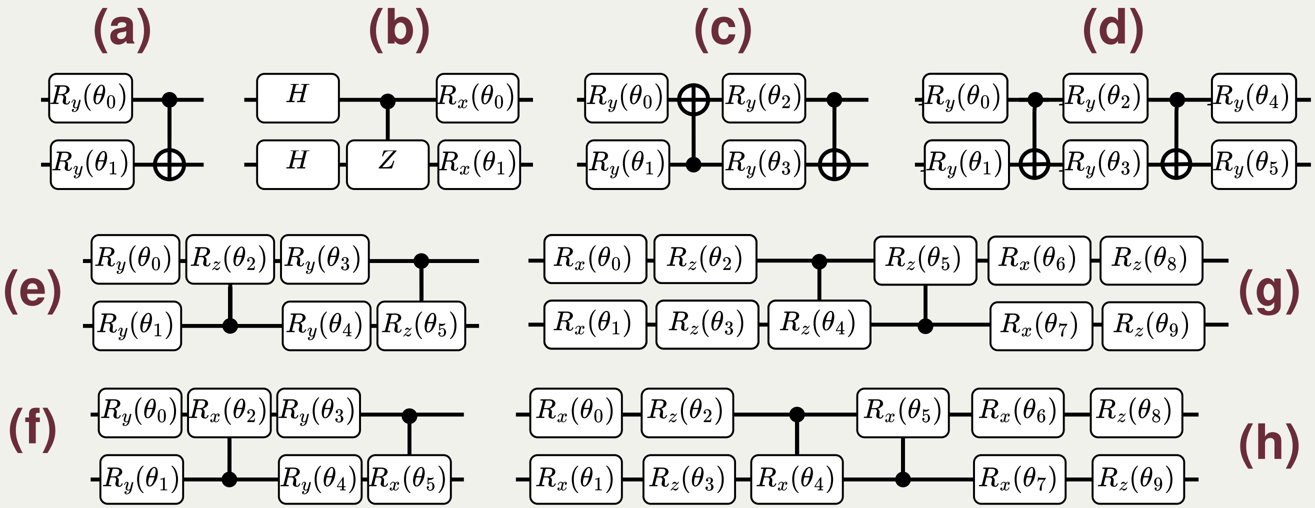
```python
# Example Search Space
strides = range(1, 8, 1)
steps = range(1, 8, 1)
mappings = [a, b, c, d, e, f, g, h]
boundaries = ["open", "periodic"]
patterns = ["10", "01", "*!", "!*", "!*!", "*!*"]

qcnn = Qinit(8) + (Qcycle(...) + Qmask(...)) * 3
```



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

Random Search

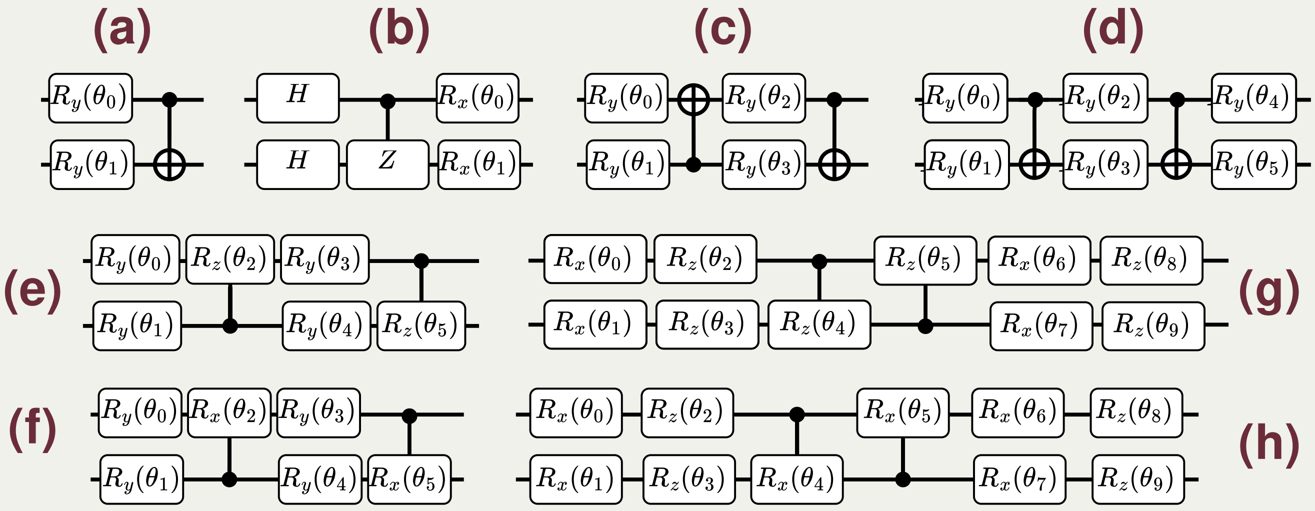
```python
# Example Search Space
strides = range(1, 8, 1)
steps = range(1, 8, 1)
mappings = [a, b, c, d, e, f, g, h]
boundaries = ["open", "periodic"]
patterns = ["10", "01", "*!", "!*", "!*!", "*!*"]

qcnn = Qinit(8) + (Qcycle(...) + Qmask(...)) * 3
```



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

Random Search

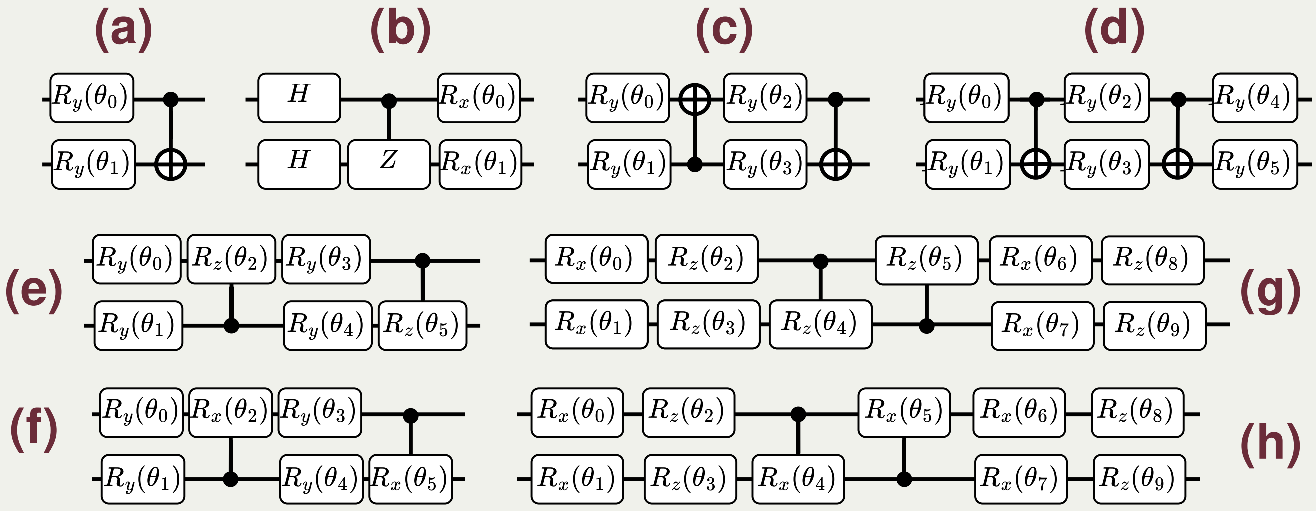
```python
# Example Search Space
strides = range(1, 8, 1)
steps = range(1, 8, 1)
mappings = [a, b, c, d, e, f, g, h]
boundaries = ["open", "periodic"]
patterns = ["10", "01", "*!", "!*", "!*!", "*!*"]

qcnn = Qinit(8) + (Qcycle(...) + Qmask(...)) * 3
```



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

Random Search

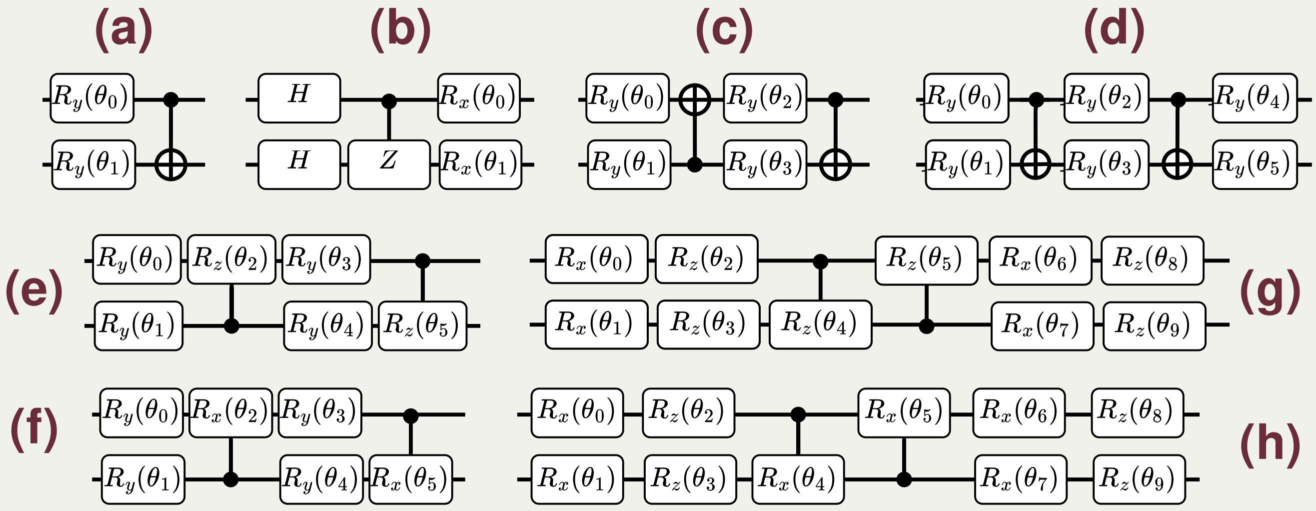
```python
# Example Search Space
strides = range(1, 8, 1)
steps = range(1, 8, 1)
mappings = [a, b, c, d, e, f, g, h]
boundaries = ["open", "periodic"]
patterns = ["10", "01", "*!", "!*", "!*!", "*!*"]

qcnn = Qinit(8) + (Qcycle(...) + Qmask(...)) * 3
```



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

Random Search

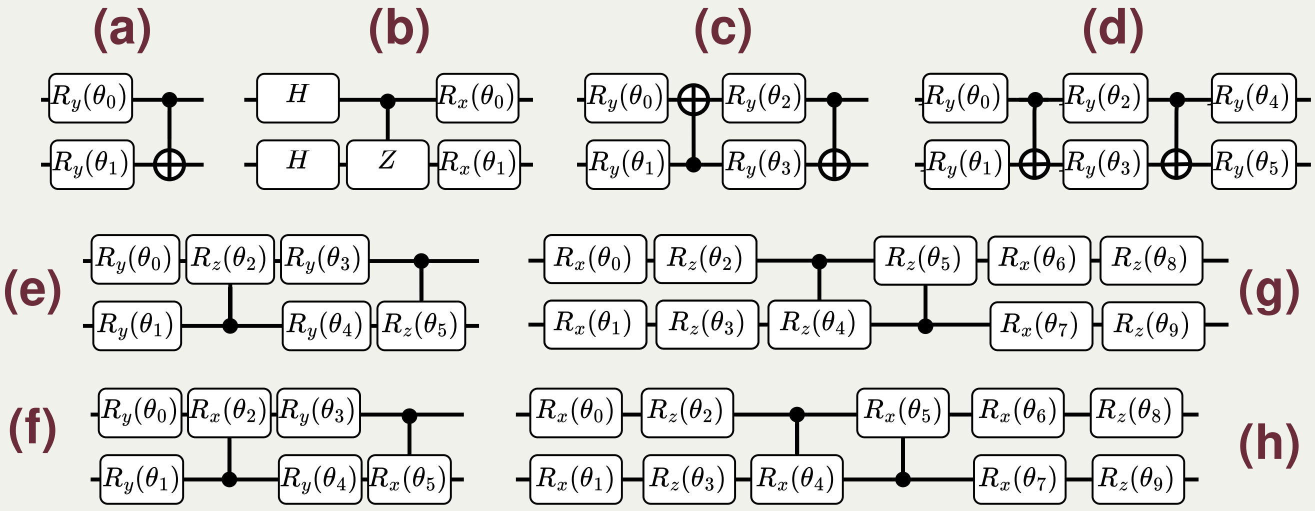
```python
# Example Search Space
strides = range(1, 8, 1)
steps = range(1, 8, 1)
mappings = [a, b, c, d, e, f, g, h]
boundaries = ["open", "periodic"]
patterns = ["10", "01", "*!", "!*", "!*!", "*!*"]

qcnn = Qinit(8) + (Qcycle(...) + Qmask(...)) * 3
```



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

Random Search

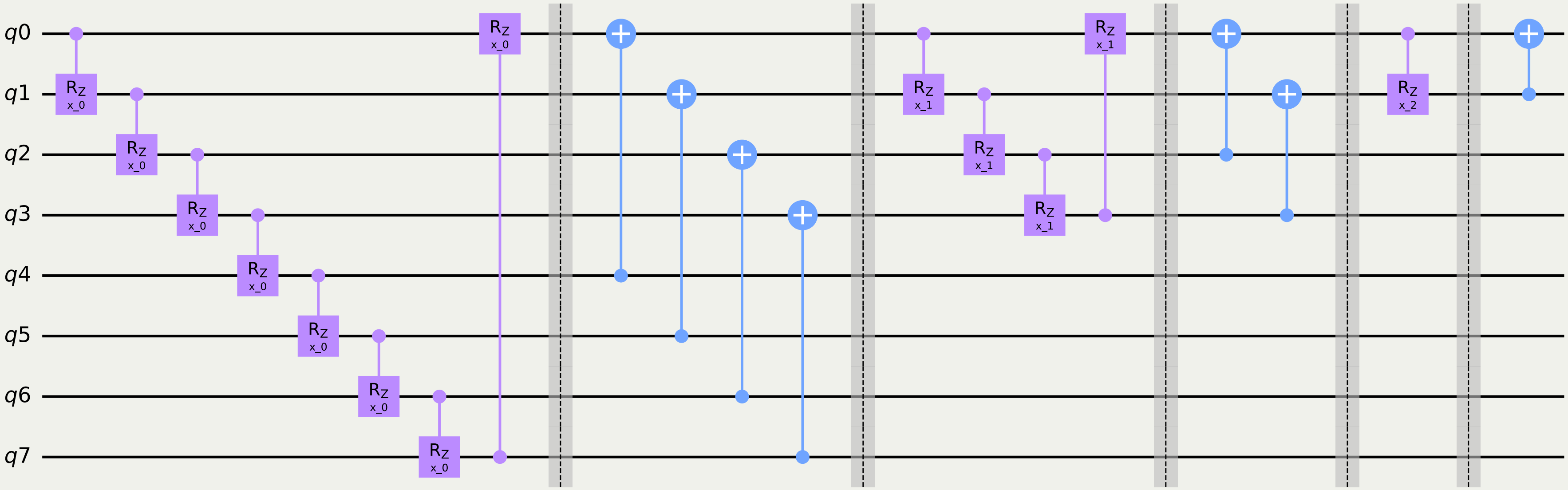
```python
# Example Search Space
strides = range(1, 8, 1)
steps = range(1, 8, 1)
mappings = [a, b, c, d, e, f, g, h]
boundaries = ["open", "periodic"]
patterns = ["10", "01", "*!", "!*", "!*!", "*!*"]

qcnn = Qinit(8) + (Qcycle(...) + Qmask(...)) * 3
```



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

Examples

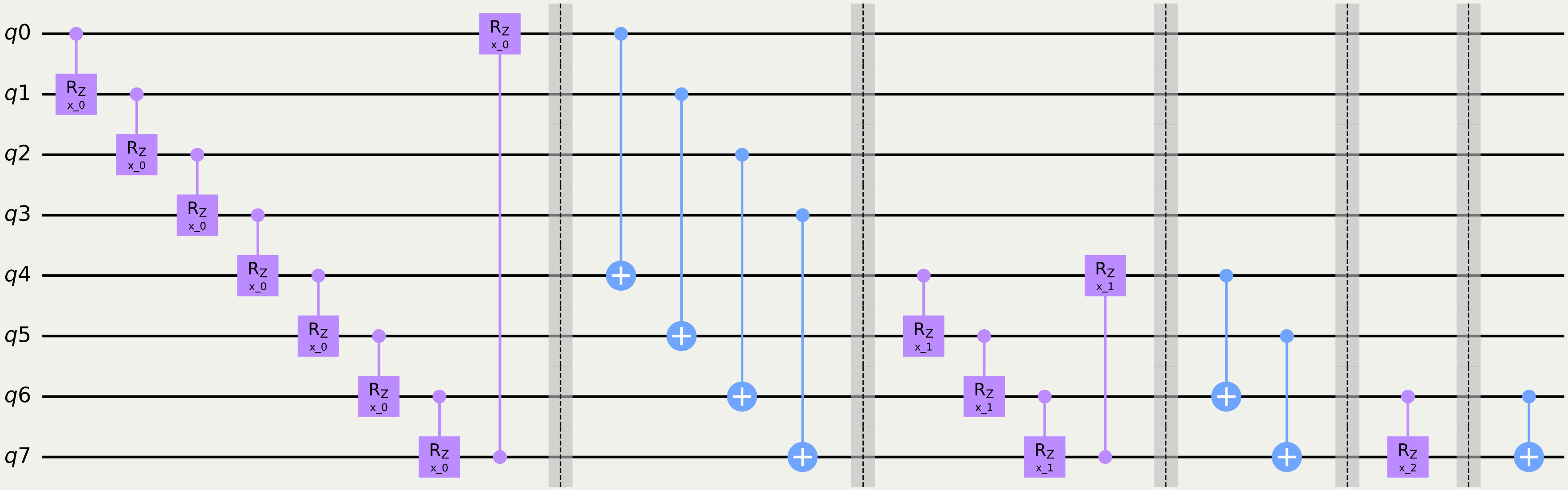
```
cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("*!", mapping=v2)
cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("!*", mapping=v2)
cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("!*!", mapping=v2)
cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("*!*", mapping=v2)
hierq = Qinit(8) + cycle_mask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



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

Examples

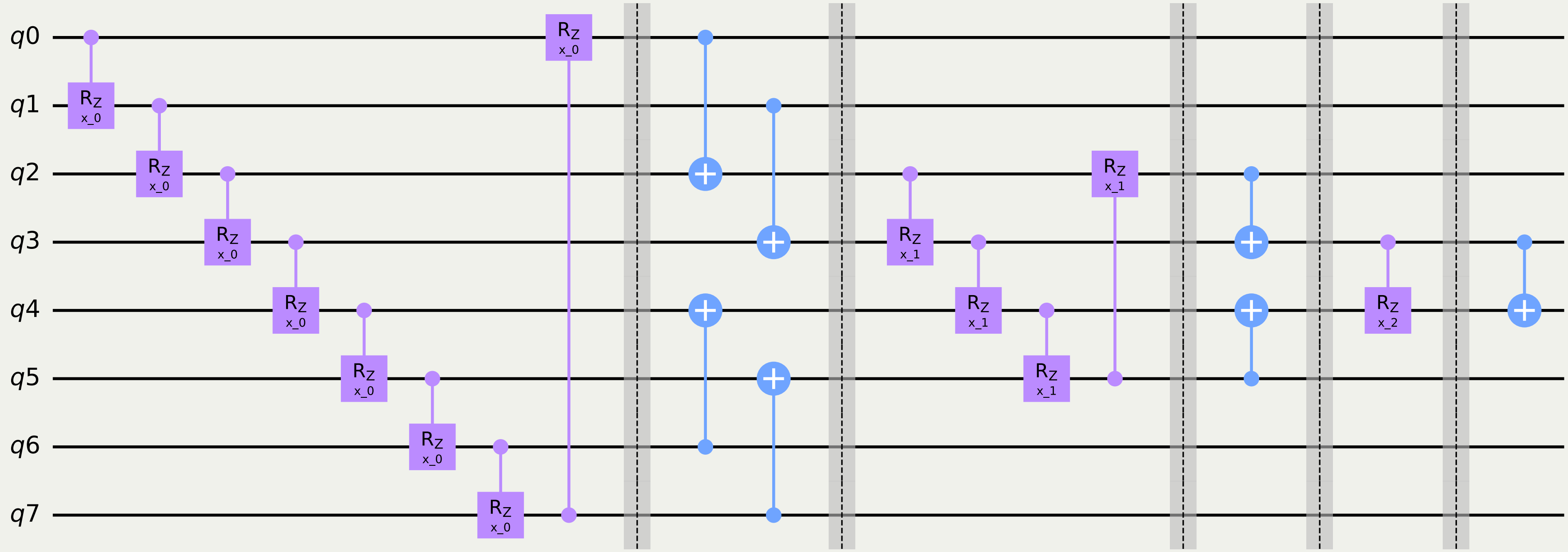
```
cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("*!", mapping=v2)
cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("!*", mapping=v2)
cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("!*!", mapping=v2)
cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("*!*", mapping=v2)
hierq = Qinit(8) + cycle_mask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



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

Examples

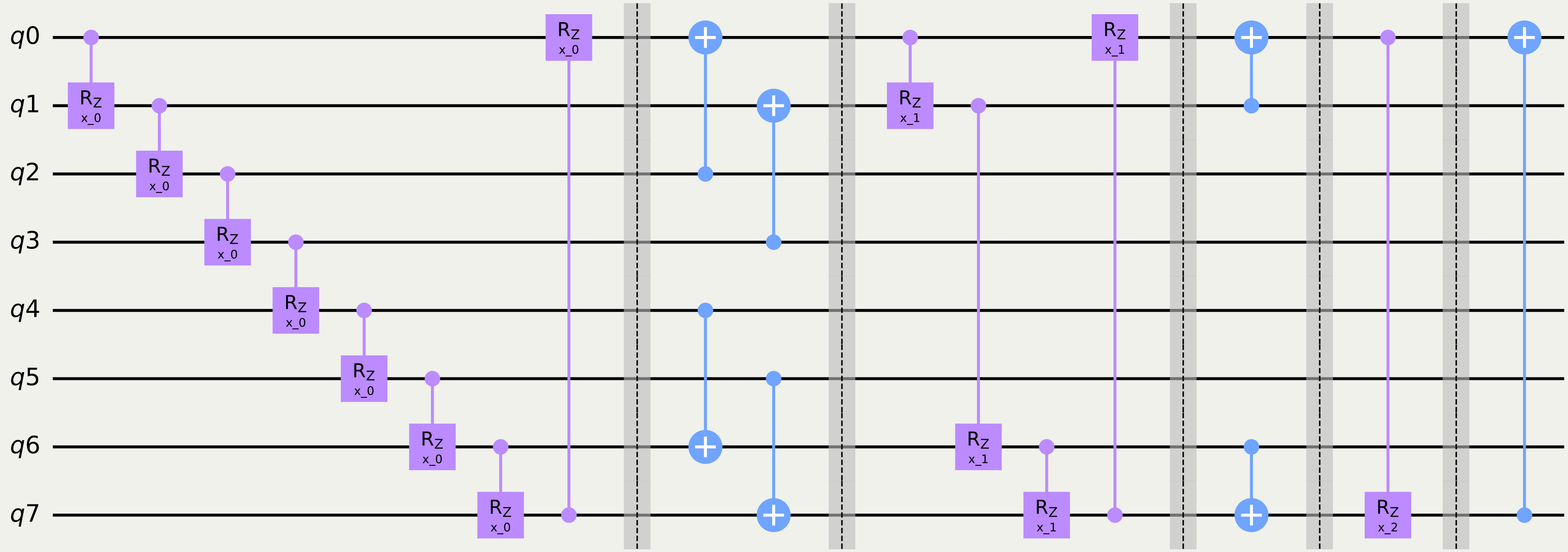
```python
cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("*!", mapping=v2)
cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("!*", mapping=v2)
cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("!*!", mapping=v2)
cycle_mask = Qcycle(1, 1, 0, mapping=u2) + Qmask("*!*", mapping=v2)
hierq = Qinit(8) + cycle_mask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



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

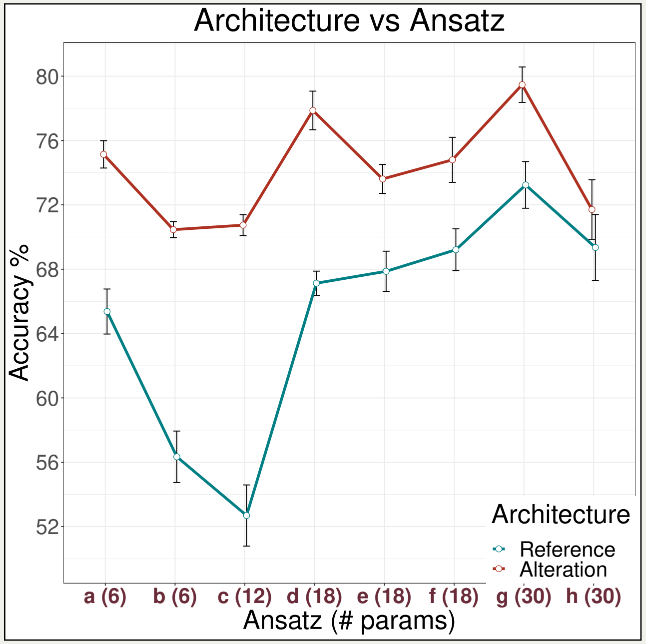
Search space

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



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

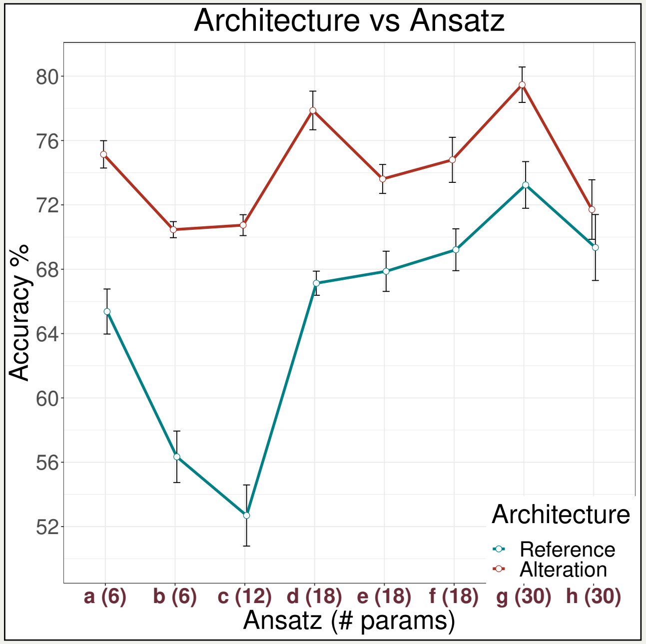
Music Genre Classification



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).

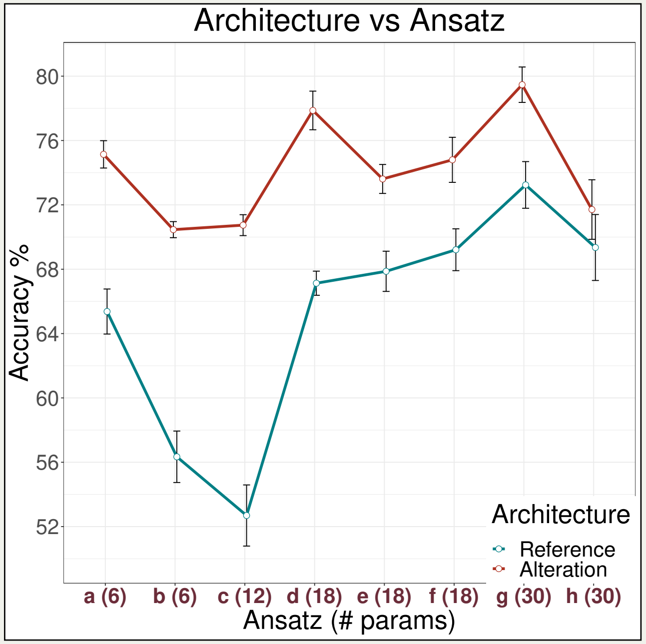
Music Genre Classification



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

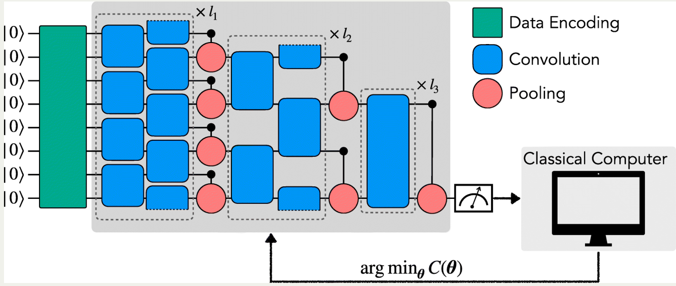
Lourens, M., Sinayskiy, I., Park, D.K. et al. Hierarchical quantum circuit representations for neural architecture search. npj Quantum Inf 9, 79 (2023).
*Hur, T., Kim, L. & Park, D. K. Quantum convolutional neural network for classical data classification. Quantum Mach. Intell. 4, 3 (2022).

Music Genre Classification



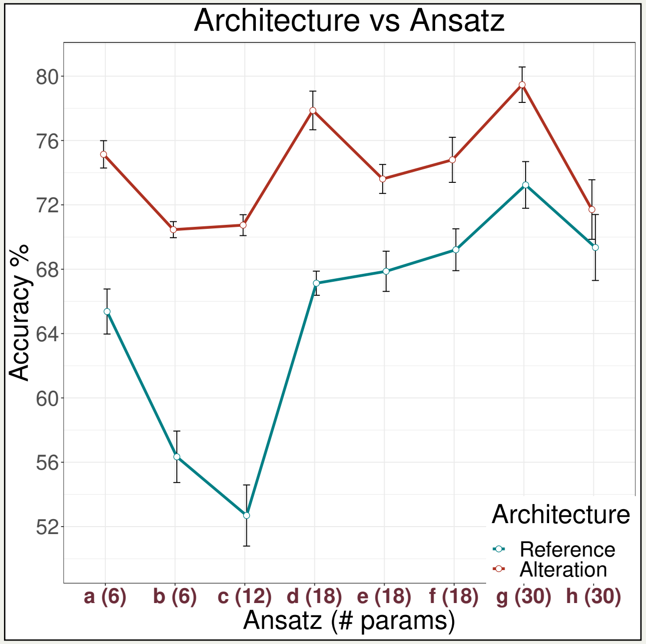
Architecture vs Ansatz				
Ansatz, # Params	Architecture: Reference	Architecture: New alteration	Δ	Alteration (s_c, F^*, s_p)
a, 6	65.37 ± 2.8	$\mathbf{75.14} \pm 1.7$	$+9.77$	$(6, \text{left}, 2)$
b, 6	56.34 ± 3.2	$\mathbf{70.46} \pm 1.0$	$+14.12$	$(1, \text{odd}, 3)$
c, 12	52.69 ± 3.8	$\mathbf{70.74}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

*Reference Architecture:



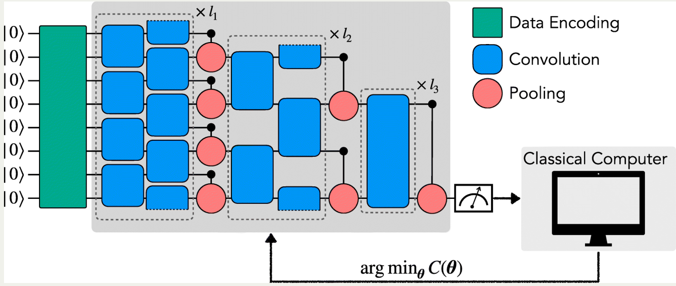
Lourens, M., Sinayskiy, I., Park, D.K. et al. Hierarchical quantum circuit representations for neural architecture search. npj Quantum Inf 9, 79 (2023).
*Hur, T., Kim, L. & Park, D. K. Quantum convolutional neural network for classical data classification. Quantum Mach. Intell. 4, 3 (2022).

Music Genre Classification

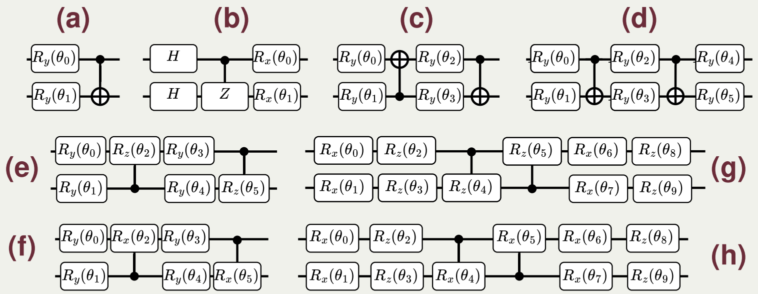


Architecture vs Ansatz				
Ansatz, # Params	Architecture: Reference	Architecture: New alteration	Δ	Alteration (s_c, F^*, s_p)
a, 6	65.37 ± 2.8	$\mathbf{75.14} \pm 1.7$	$+9.77$	$(6, \text{left}, 2)$
b, 6	56.34 ± 3.2	$\mathbf{70.46} \pm 1.0$	$+14.12$	$(1, \text{odd}, 3)$
c, 12	52.69 ± 3.8	$\mathbf{70.74}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

*Reference Architecture:



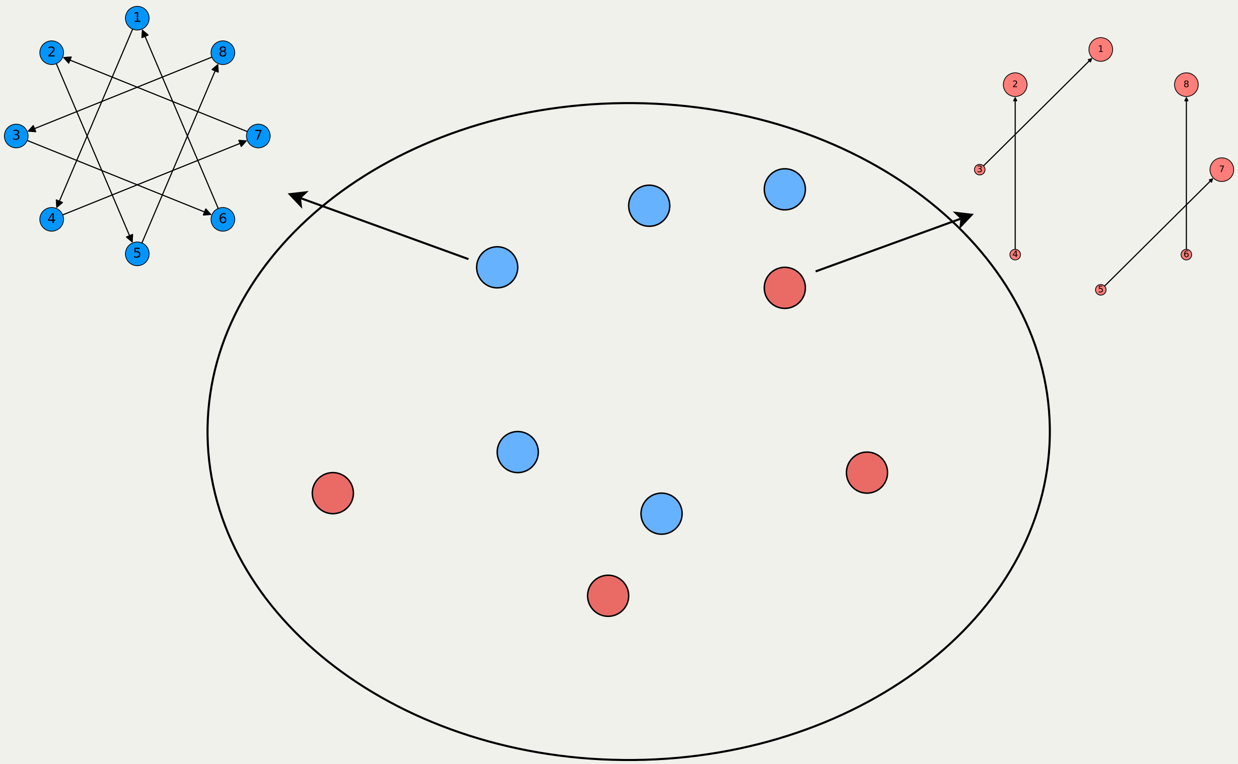
Ansatz:



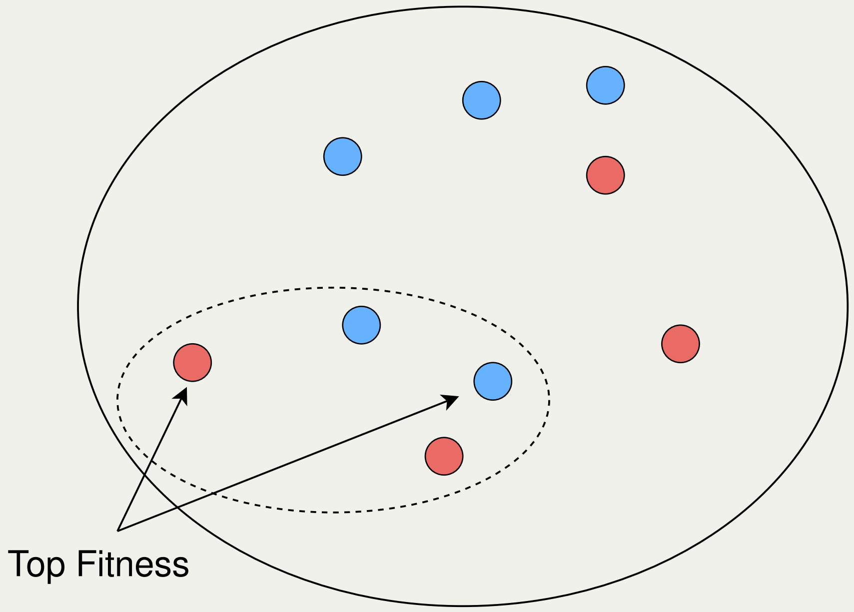
Lourens, M., Sinayskiy, I., Park, D.K. et al. Hierarchical quantum circuit representations for neural architecture search. npj Quantum Inf 9, 79 (2023).
*Hur, T., Kim, L. & Park, D. K. Quantum convolutional neural network for classical data classification. Quantum Mach. Intell. 4, 3 (2022).

Evolutionary Search

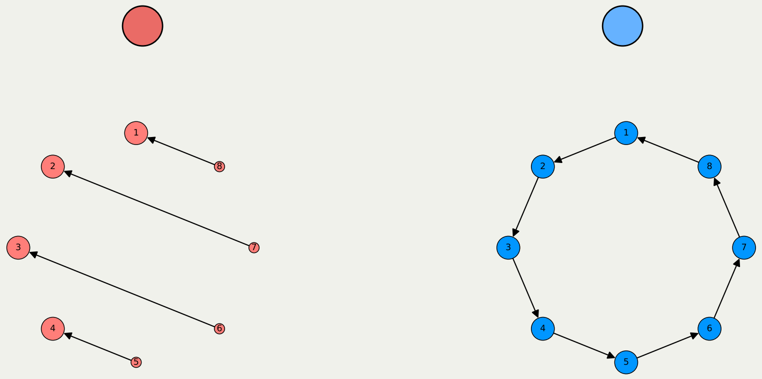
Evolutionary Search



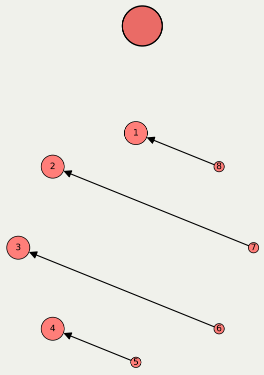
Tournament Selection

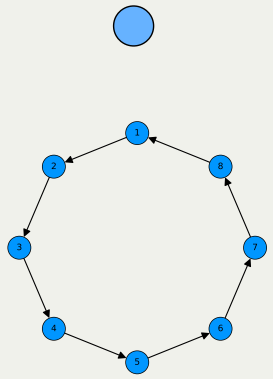


Tournament Selection

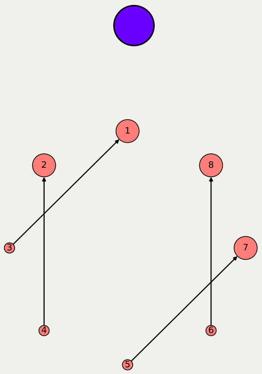


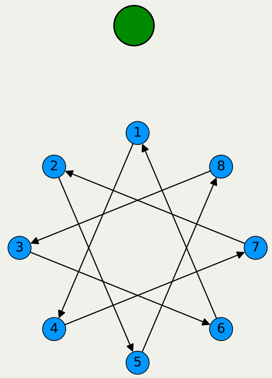
Tournament Selection



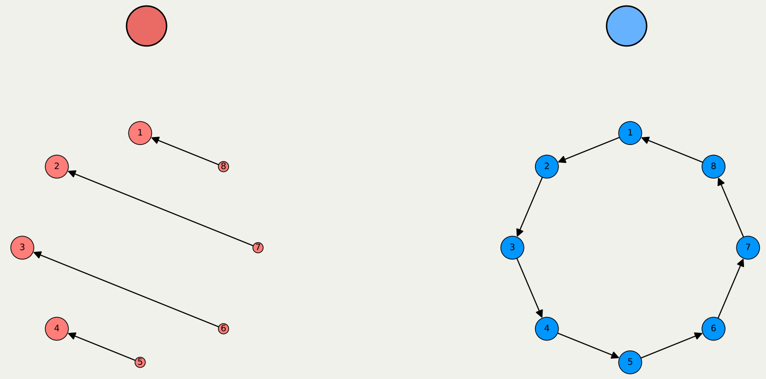


Mutation

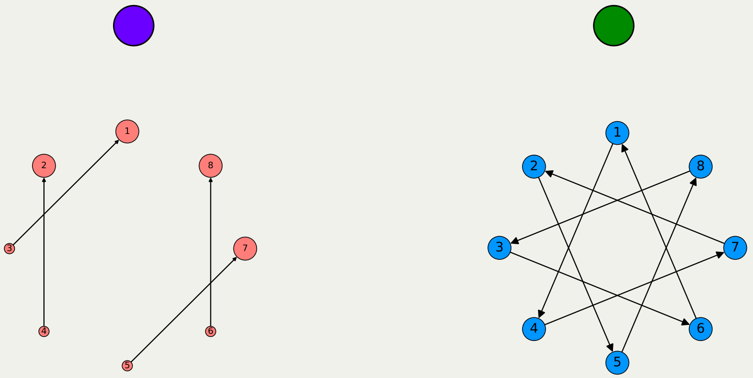




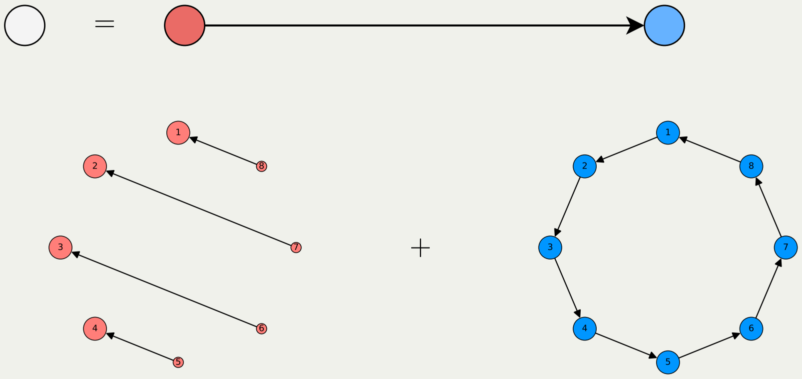
Tournament Selection



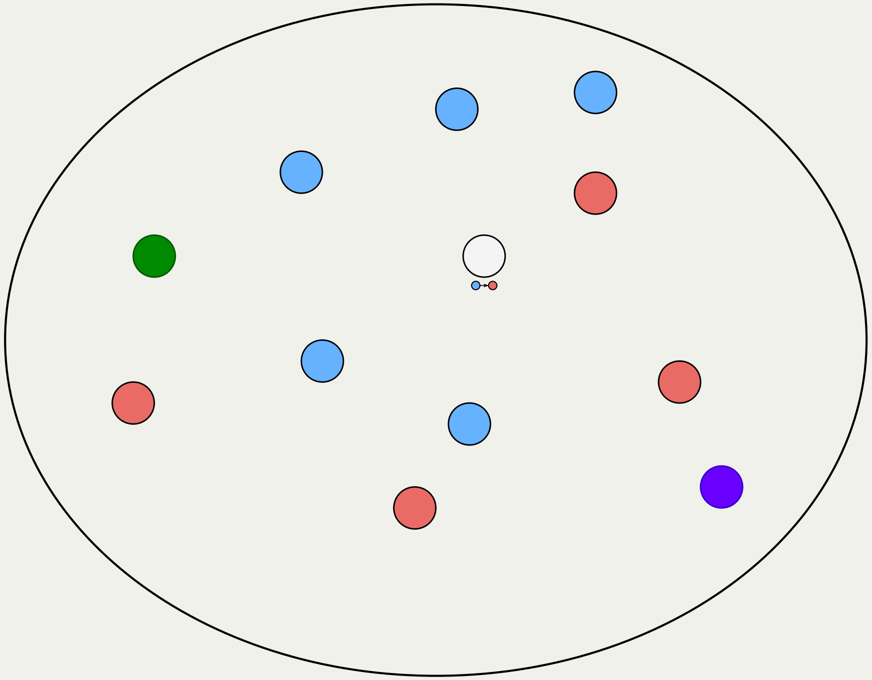
Mutation



Crossover



Repeat



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:

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

- Given an unknown ground state $|\psi_g\rangle$, what is the phase?

Quantum Phase Recognition

Cluster-Ising Hamiltonian:

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

- Given an unknown ground state $|\psi_g\rangle$, what is the phase?
  - Paramagnetic

Quantum Phase Recognition

Cluster-Ising Hamiltonian:

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

- Given an unknown ground state $|\psi_g\rangle$, what is the phase?
  - Paramagnetic
  - $\mathbb{Z}_2 \times \mathbb{Z}_2$ Symmetry-Protected Topological (SPT)

Quantum Phase Recognition

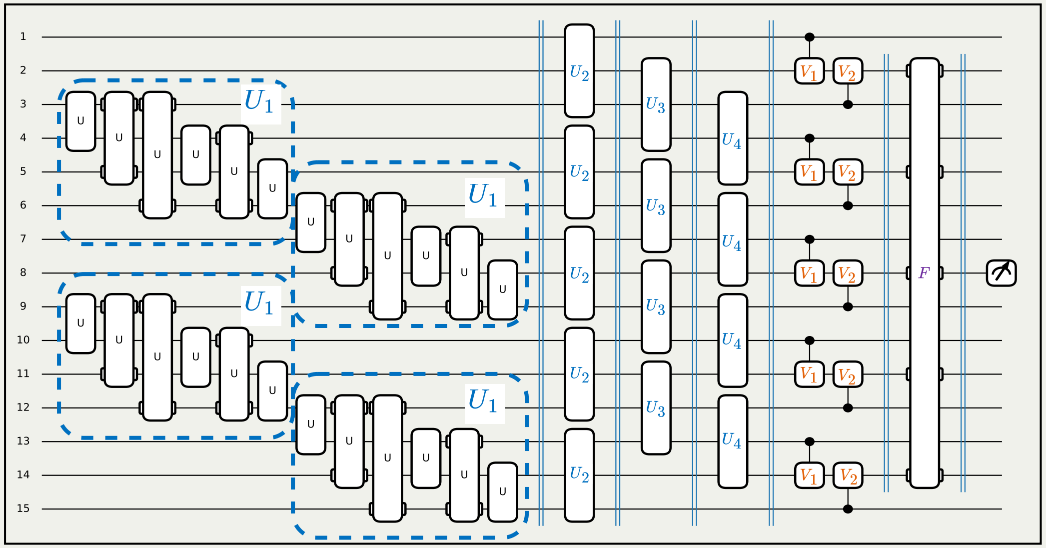
Cluster-Ising Hamiltonian:

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

- Given an unknown ground state $|\psi_g\rangle$, what is the phase?
  - Paramagnetic
  - $\mathbb{Z}_2 \times \mathbb{Z}_2$ Symmetry-Protected Topological (SPT)
  - Antiferromagnetic

Quantum Phase Recognition

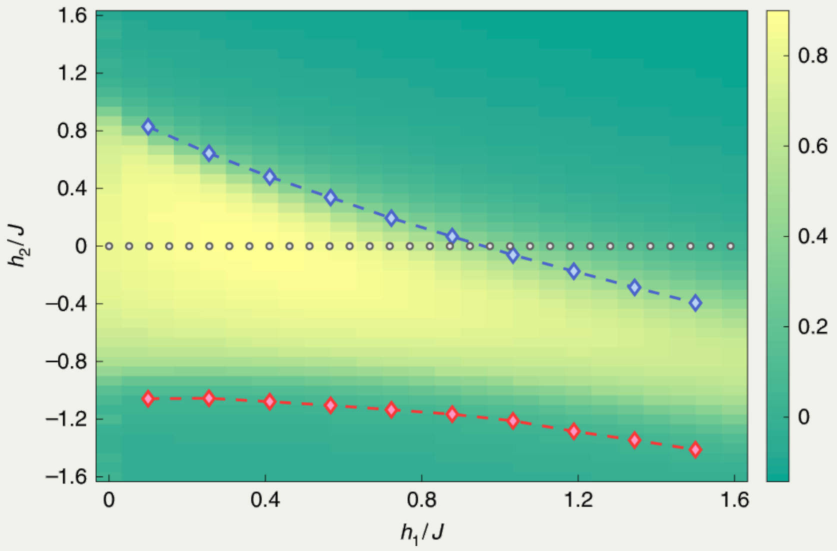
```
u0 = Qinit(4) + Qpermute(share_weights=False, mapping=Ugm)
u1 = Qcycle(1,3,2, mapping=u0, boundary="open", edge_order=[1,3,2,4])
u2 = sum([Qcycle(1,3,offset, mapping=Ugm, boundary="open") for offset in range(3)])
u3 = Qmask("101",1,3,0, mapping=Vgm, boundary="open")
u4 = Qcycle(1, mapping=Ugm, boundary="open")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

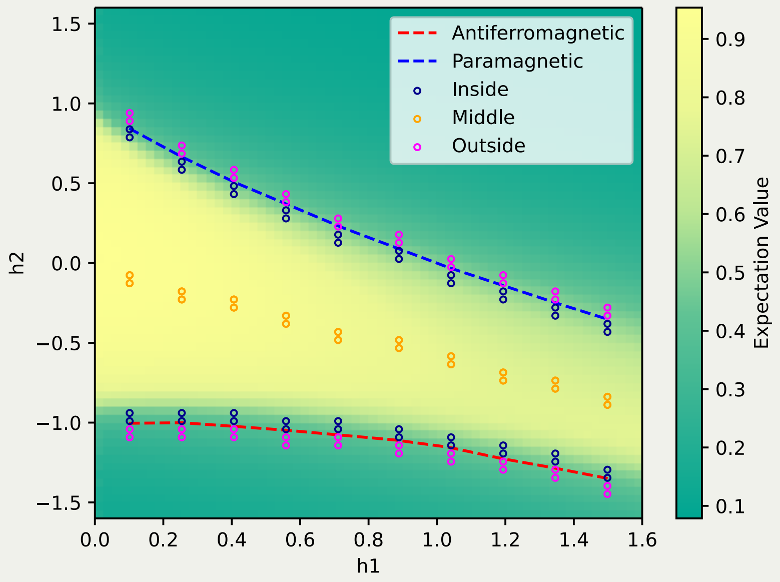
Quantum Phase Recognition

Reference[1]



→

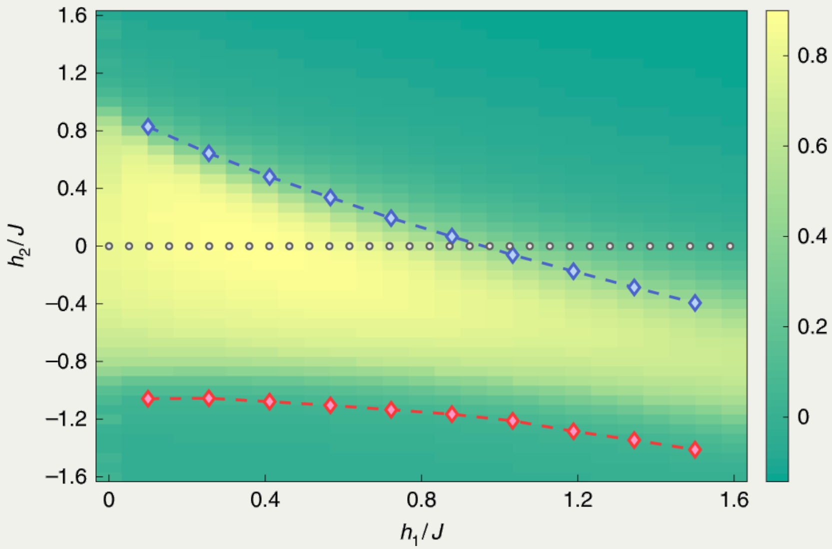
Found[2]



[1]Cong, I., Choi, S. & Lukin, M.D. Quantum convolutional neural networks. Nat. Phys. 15, 1273-1278 (2019).
[2]Lourens, M., Sinayskiy, I., Park, D.K. et al. Hierarchical quantum circuit representations for neural architecture search. npj Quantum Inf 9, 79 (2023).

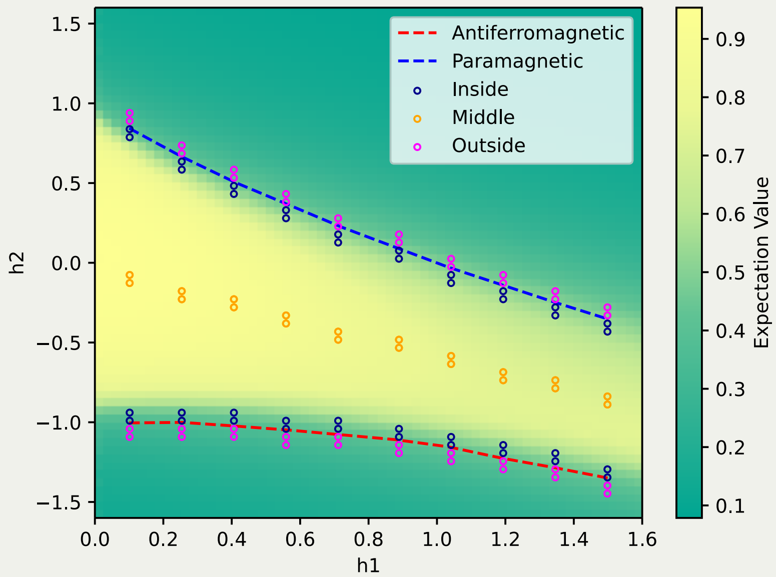
Quantum Phase Recognition

Reference[1]



→

Found[2]

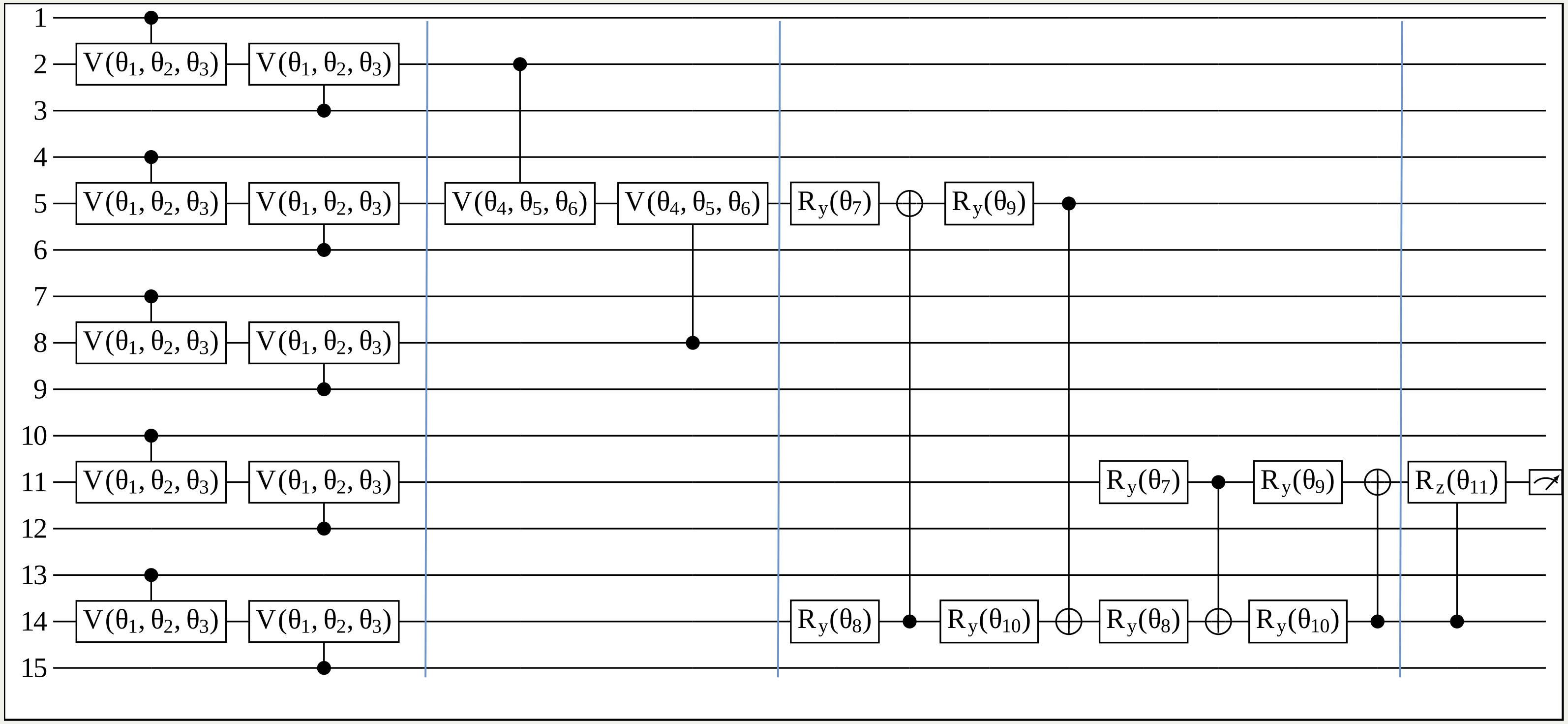


Metric	Reference	Found
Number of parameters	1308	**11**
Sample Complexity (Inside)	61.523	**36.079**
Sample Complexity (Middle)	**10.992**	13.253
MSE (Outside)	**0.164**	0.167

[1] Cong, I., Choi, S. & Lukin, M.D. Quantum convolutional neural networks. Nat. Phys. 15, 1273-1278 (2019).
[2] 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

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

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

What's next?

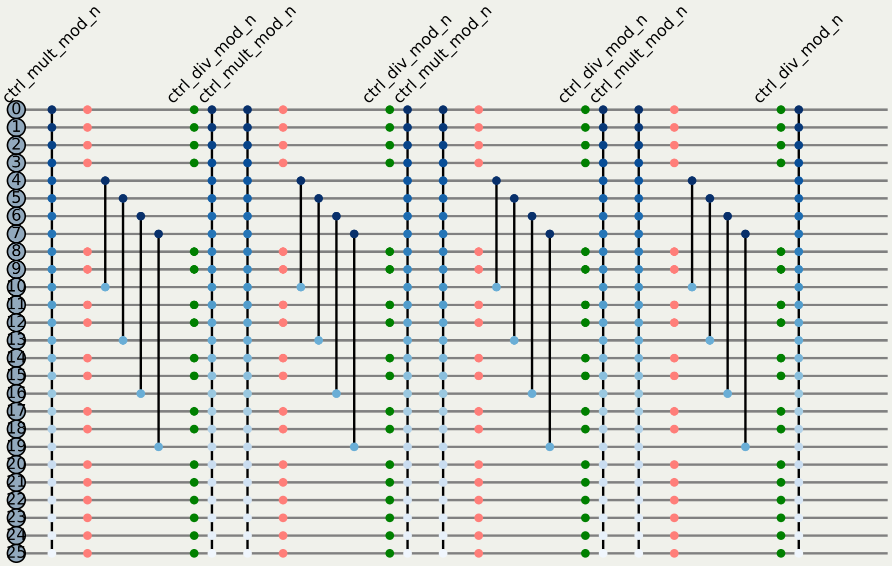
Compute Graph Design

What's next?

Compute Graph Design

- Quantum Circuits

Exponentiation modulo N



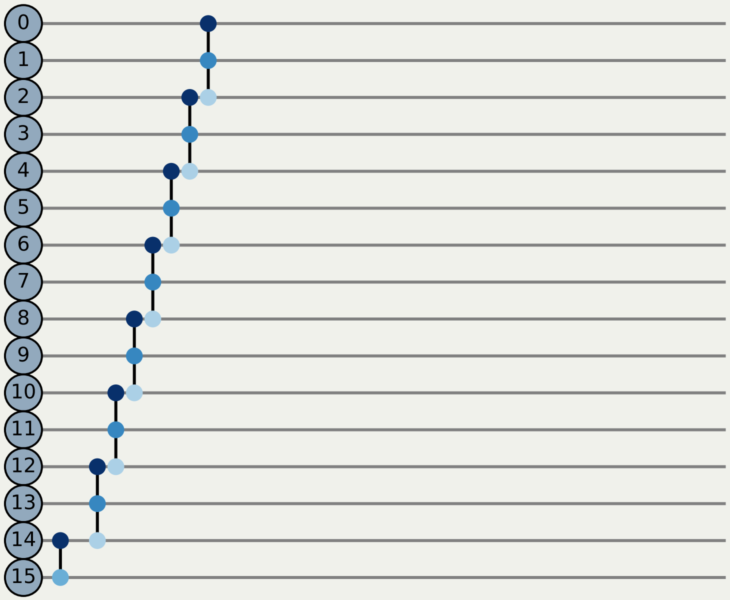
What's next?

Compute Graph Design

- Quantum Circuits
- Classical Circuits

Addition

```python
def or(bits, symbols=None, state=None):
  b1, b2 = state[bits[0]], state[bits[1]]
  state[bits[0]] = b1 or b2
  return state
```



What's next?

Compute Graph Design

- Quantum Circuits
- Classical Circuits
- Tensor Networks

```
tensors = [np.array([1, 0])] * 5
hierq = (Qinit(5, tensors=tensors)
       + mask_anc
       + U_psi + grover)
psi = hierq()
```

What's next?

Compute Graph Design

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

?

What's next?

Compute Graph Design

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

Paper

Hierarqcal

Slides

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)

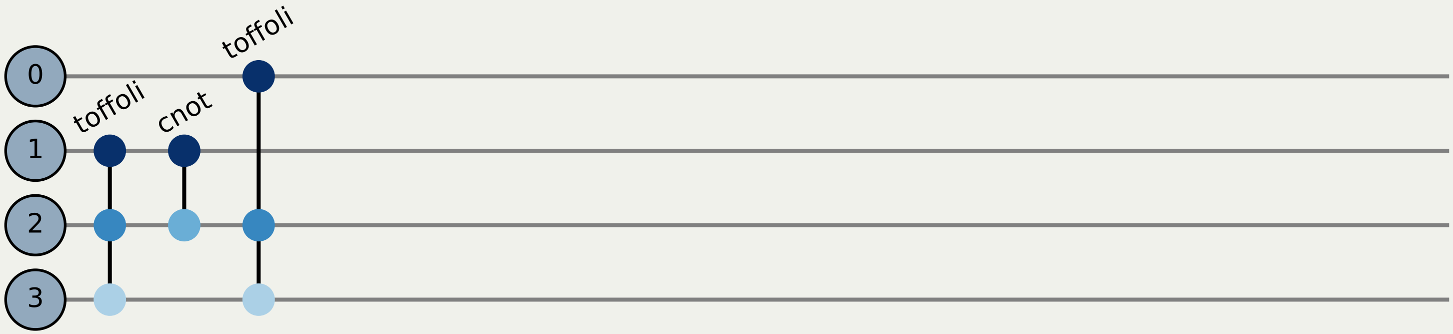
Shor's Algorithm

- 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)

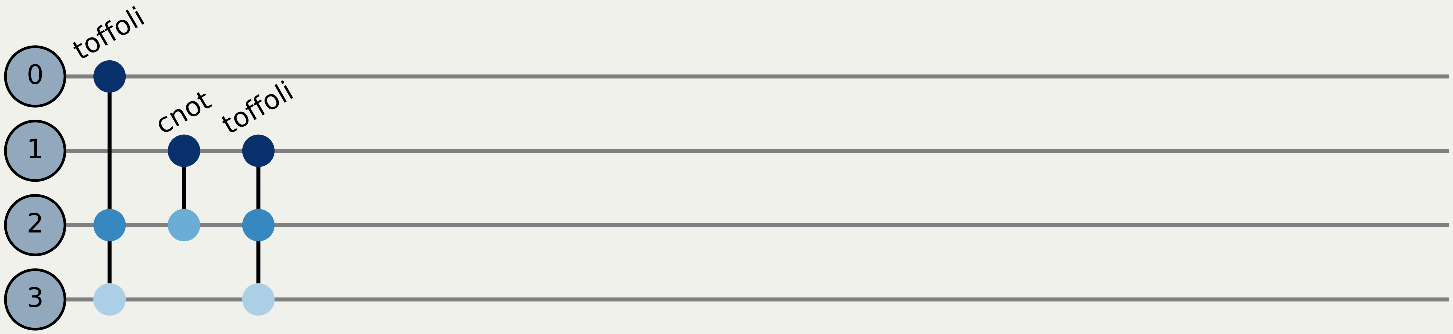
Shor circuit

```python
# ====== Motifs level 1
carry_motif = (
  Qinit(4)
  + Qmotif(E=[(1, 2, 3)], mapping=toffoli)
  + Qmotif(E=[(1, 2)], mapping=cnot)
  + Qmotif(E=[(0, 2, 3)], mapping=toffoli)
)

# Turn carry into a function to be able to reverse it easily
carry = lambda r: carry_motif if r == 1 else carry_motif.reverse()
plot_circuit(carry(1))
plot_circuit(carry(-1))
```



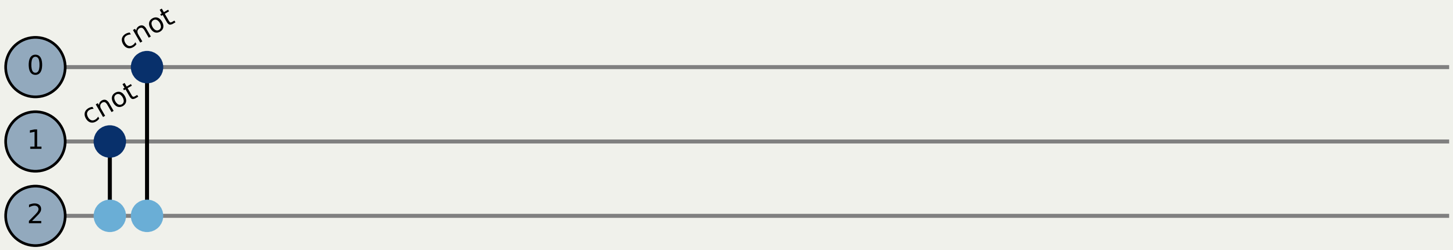
Shor circuit

```python
# ====== Motifs level 1
carry_motif = (
  Qinit(4)
  + Qmotif(E=[(1, 2, 3)], mapping=toffoli)
  + Qmotif(E=[(1, 2)], mapping=cnot)
  + Qmotif(E=[(0, 2, 3)], mapping=toffoli)
)

# Turn carry into a function to be able to reverse it easily
carry = lambda r: carry_motif if r == 1 else carry_motif.reverse()
plot_circuit(carry(1))
plot_circuit(carry(-1))
```



Shor circuit

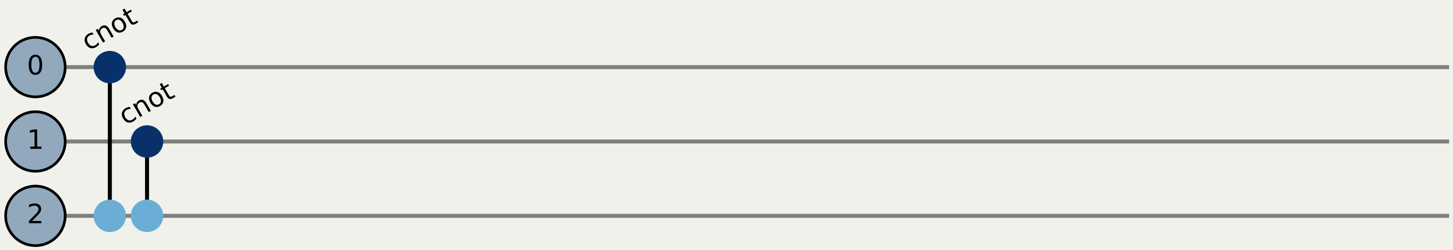
```
# ====== Motifs level 1
sum = lambda r=1: Qinit(3, name=f"sum") + Qpivot(
  "*1", merge_within="01", mapping=cnot, edge_order=[-1 * r]
)

plot_circuit(sum(1))
plot_circuit(sum(-1))
```



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

Shor circuit

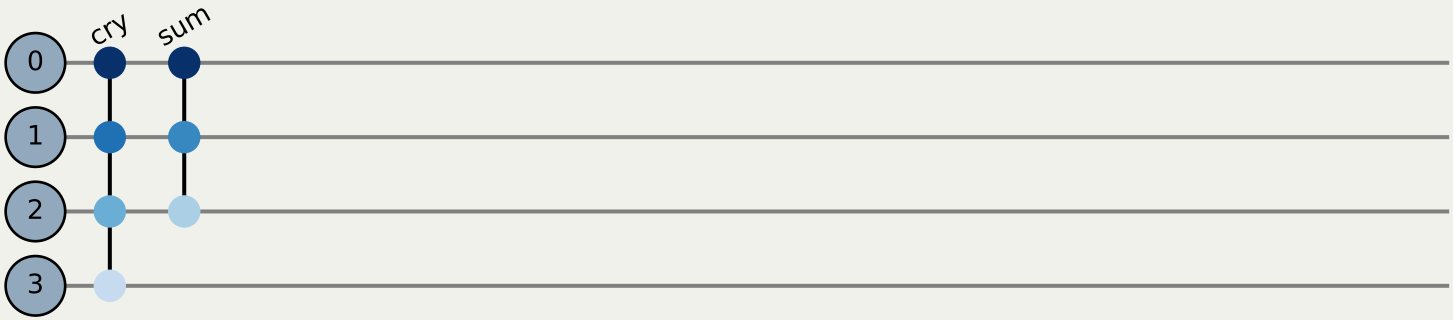
```
# ====== Motifs level 1
sum = lambda r=1: Qinit(3, name=f"sum") + Qpivot(
  "*1", merge_within="01", mapping=cnot, edge_order=[-1 * r]
)

plot_circuit(sum(1))
plot_circuit(sum(-1))
```



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

Shor circuit

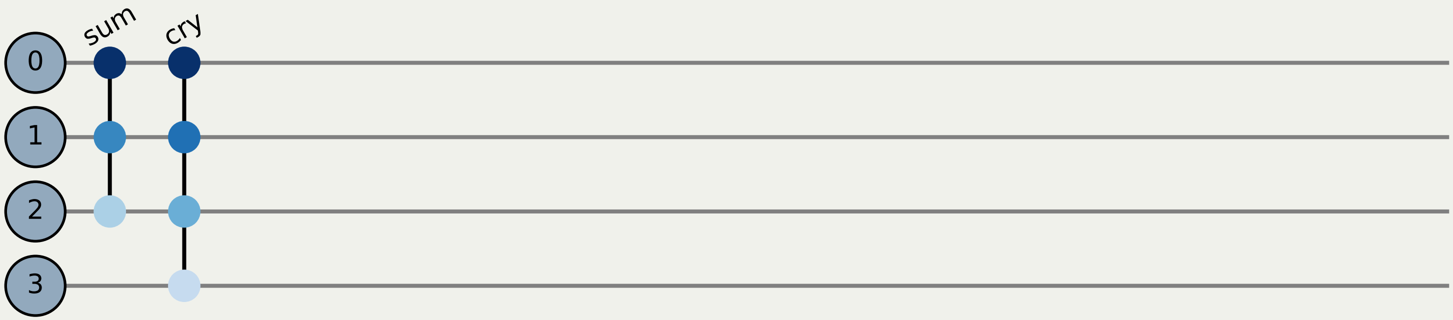
```
# ====== Motifs level 2
carry_sum_motif = (
  lambda r=1: Qinit(4, name=f"crs")
  + Qpivot("1*", merge_within="1111", mapping=carry(-r))
  + Qpivot("1*", merge_within="111", mapping=sum(r))
)

carry_sum = lambda r=1: carry_sum_motif(1) if r == 1 else carry_sum_motif(-1).reverse

plot_circuit(carry_sum(1))
plot_circuit(carry_sum(-1))
```



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

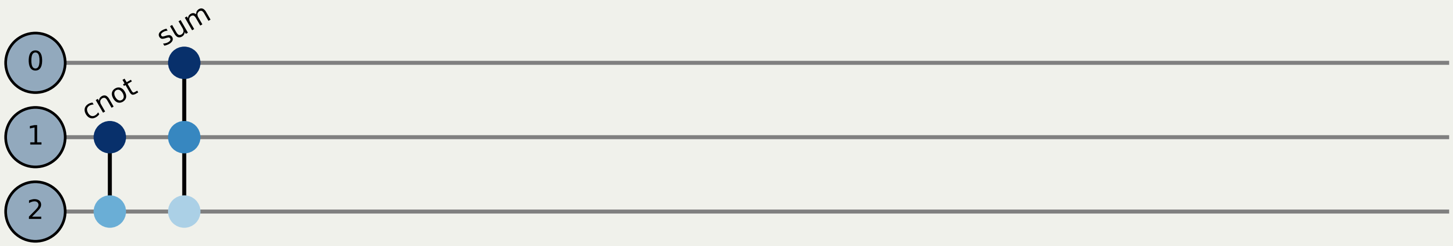
Shor circuit

```
# ====== Motifs level 2
carry_sum_motif = (
  lambda r=1: Qinit(4, name=f"crs")
  + Qpivot("1*", merge_within="1111", mapping=carry(-r))
  + Qpivot("1*", merge_within="111", mapping=sum(r))
)

carry_sum = lambda r=1: carry_sum_motif(1) if r == 1 else carry_sum_motif(-1).reverse

plot_circuit(carry_sum(1))
plot_circuit(carry_sum(-1))
```



Shor circuit

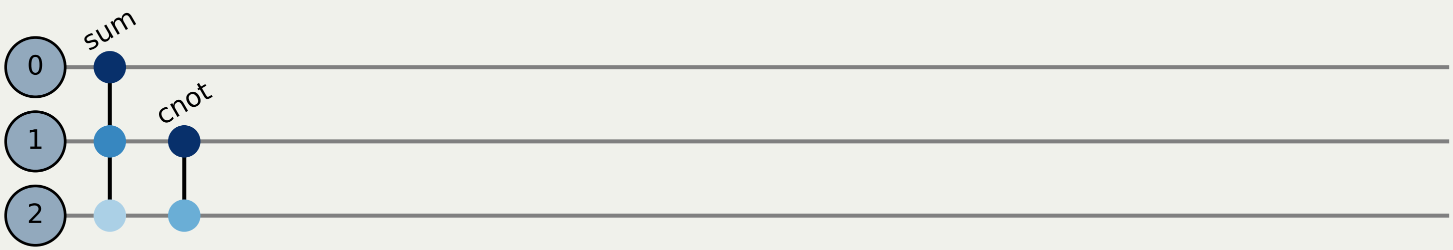
```
cnot_sum_motif = (
  lambda r=1: Qinit(3, name=f"cns")
  + Qpivot("*1", merge_within="11", mapping=cnot)
  + Qpivot("*1", merge_within="111", mapping=sum(r))
)

cnot_sum = lambda r=1: cnot_sum_motif(1) if r == 1 else cnot_sum_motif(-1).reverse()

plot_circuit(cnot_sum(1))
plot_circuit(cnot_sum(-1))
```



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

Shor circuit

```
cnot_sum_motif = (
  lambda r=1: Qinit(3, name=f"cns")
  + Qpivot("*1", merge_within="11", mapping=cnot)
  + Qpivot("*1", merge_within="111", mapping=sum(r))
)

cnot_sum = lambda r=1: cnot_sum_motif(1) if r == 1 else cnot_sum_motif(-1).reverse()

plot_circuit(cnot_sum(1))
plot_circuit(cnot_sum(-1))
```



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

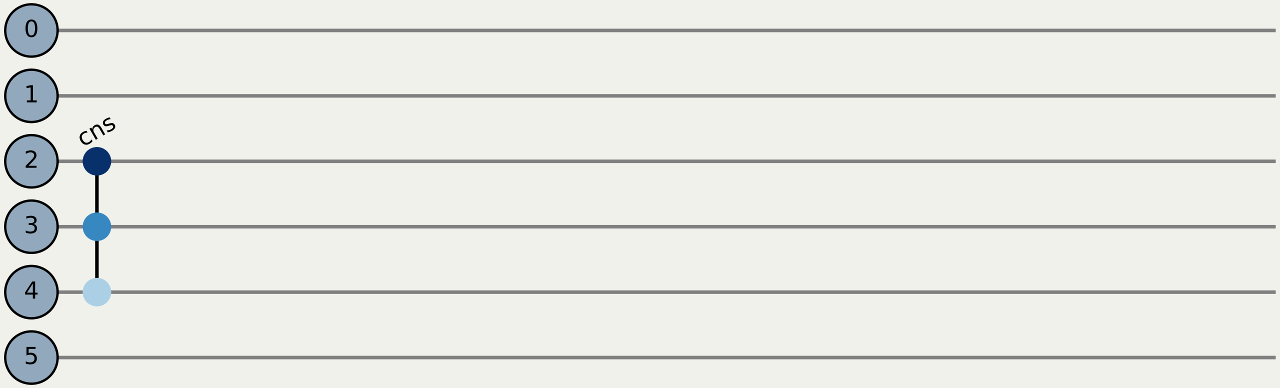
Shor circuit

```python
# ====== Motifs level 3
carry_layer = lambda r=1: Qcycle(
  1,
  3,
  0,
  mapping=carry(r),
  boundary="open",
  edge_order=[r],
)
plot_circuit(carry_layer_1)
```

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

Shor circuit

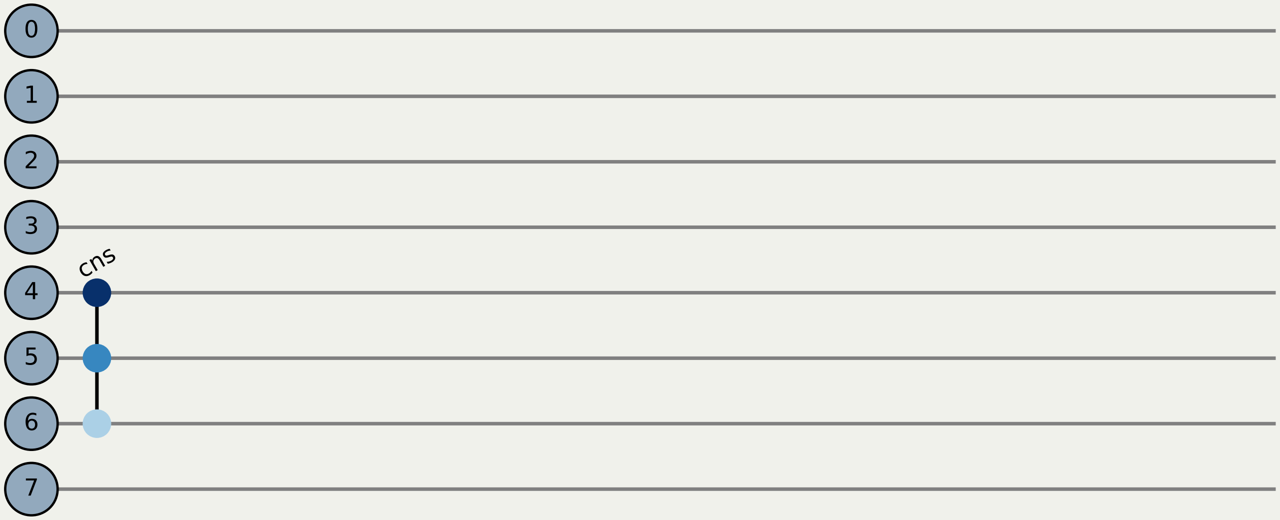
```
cnot_sum_pivot = lambda r: Qpivot("*10", merge_within="111", mapping=cnot_sum(r))

plot_circuit(Qinit(6) + cnot_sum_pivot(1))
plot_circuit(Qinit(8) + cnot_sum_pivot(1))
```



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

Shor circuit

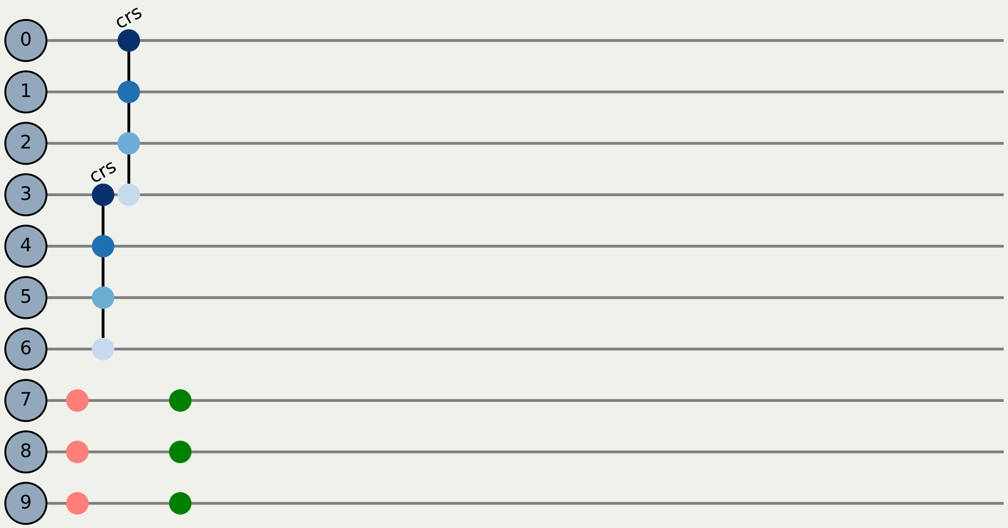
```
cnot_sum_pivot = lambda r: Qpivot("*10", merge_within="111", mapping=cnot_sum(r))

plot_circuit(Qinit(6) + cnot_sum_pivot(1))
plot_circuit(Qinit(8) + cnot_sum_pivot(1))
```



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

Shor circuit

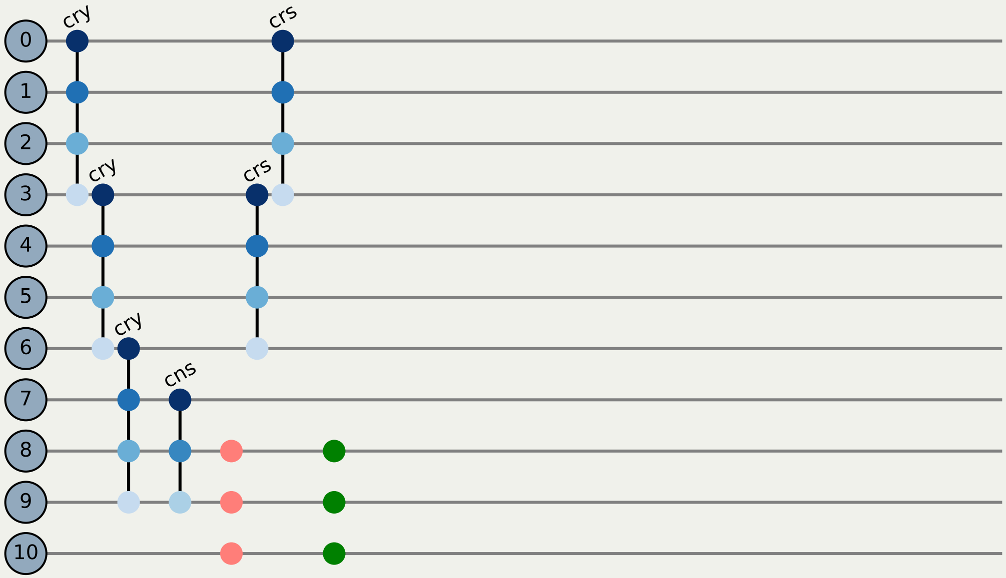
```
carry_sum_layer = lambda r: (
  Qmask("*111")
  + Qcycle(1, 3, 0,
    boundary="open",
    mapping=carry_sum(r),
    edge_order=[-r],
  )
  + Qunmask("previous")
)
plot_circuit(Qinit(10) + carry_sum_layer(1))
```



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

Shor circuit

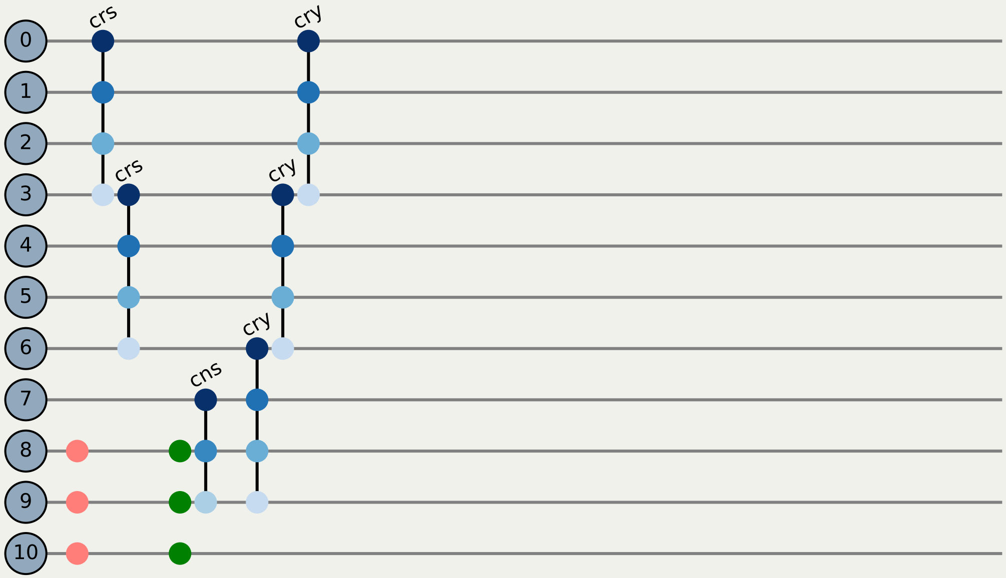
```
addition = carry_layer(1) + cnot_sum_pivot(1) + carry_sum_layer(1)
subtraction = carry_sum_layer(-1) + cnot_sum_pivot(-1) + carry_layer(-1)

plot_circuit(Qinit(11) + addition)
plot_circuit(Qinit(11) + subtraction)
plot_circuit(Qinit(13) + addition)
plot_circuit(Qinit(13) + subtraction)
```



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

Shor circuit

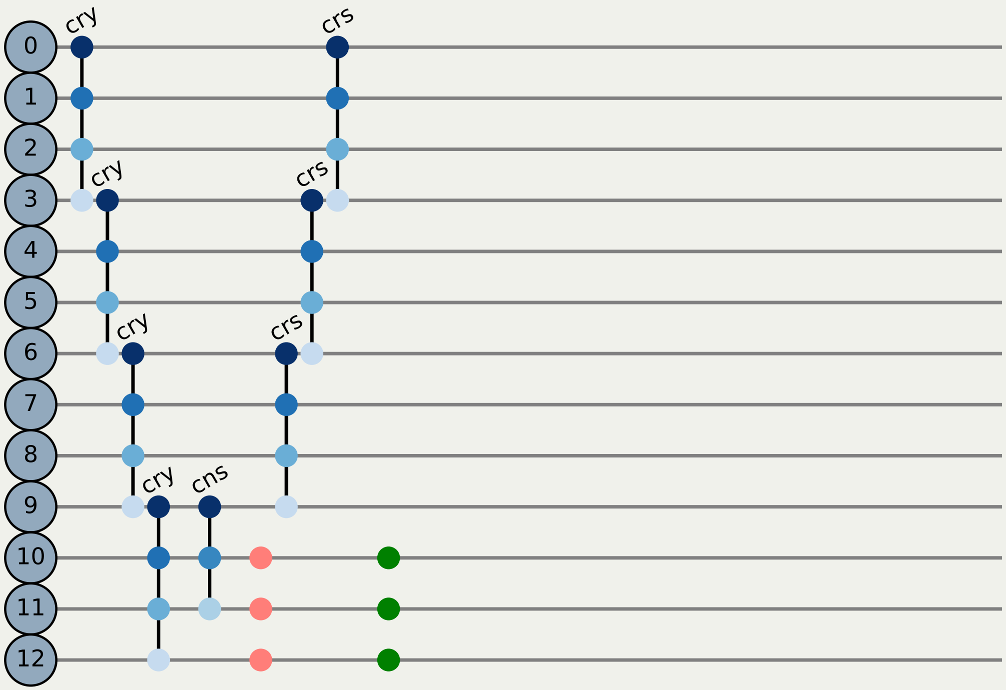
```
addition = carry_layer(1) + cnot_sum_pivot(1) + carry_sum_layer(1)
subtraction = carry_sum_layer(-1) + cnot_sum_pivot(-1) + carry_layer(-1)

plot_circuit(Qinit(11) + addition)
plot_circuit(Qinit(11) + subtraction)
plot_circuit(Qinit(13) + addition)
plot_circuit(Qinit(13) + subtraction)
```



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

Shor circuit

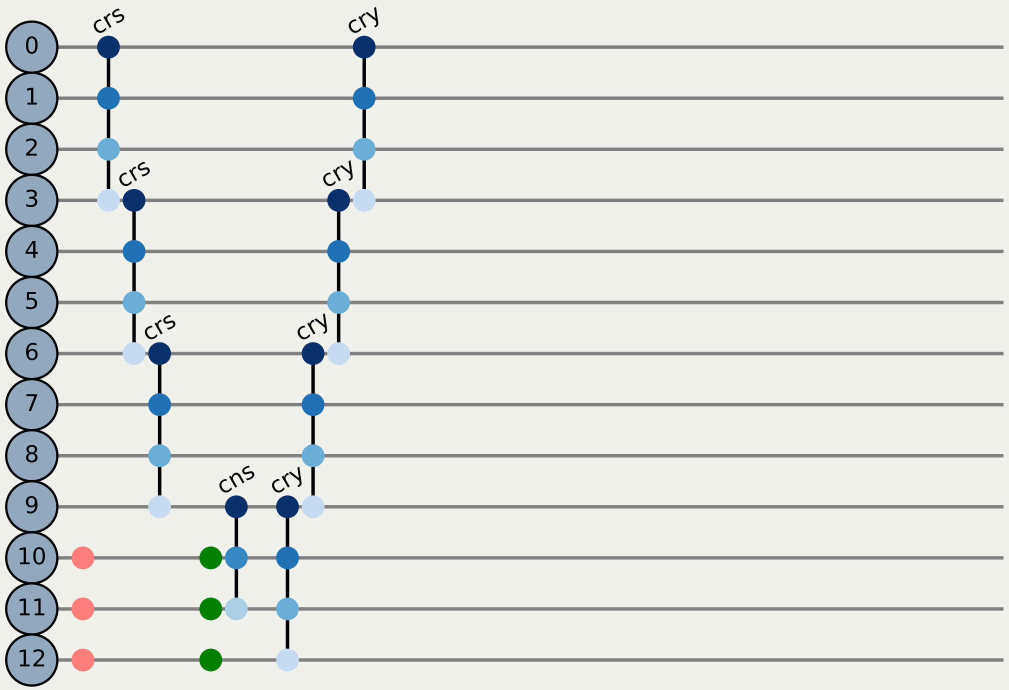
```
addition = carry_layer(1) + cnot_sum_pivot(1) + carry_sum_layer(1)
subtraction = carry_sum_layer(-1) + cnot_sum_pivot(-1) + carry_layer(-1)

plot_circuit(Qinit(11) + addition)
plot_circuit(Qinit(11) + subtraction)
plot_circuit(Qinit(13) + addition)
plot_circuit(Qinit(13) + subtraction)
```



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

Shor circuit

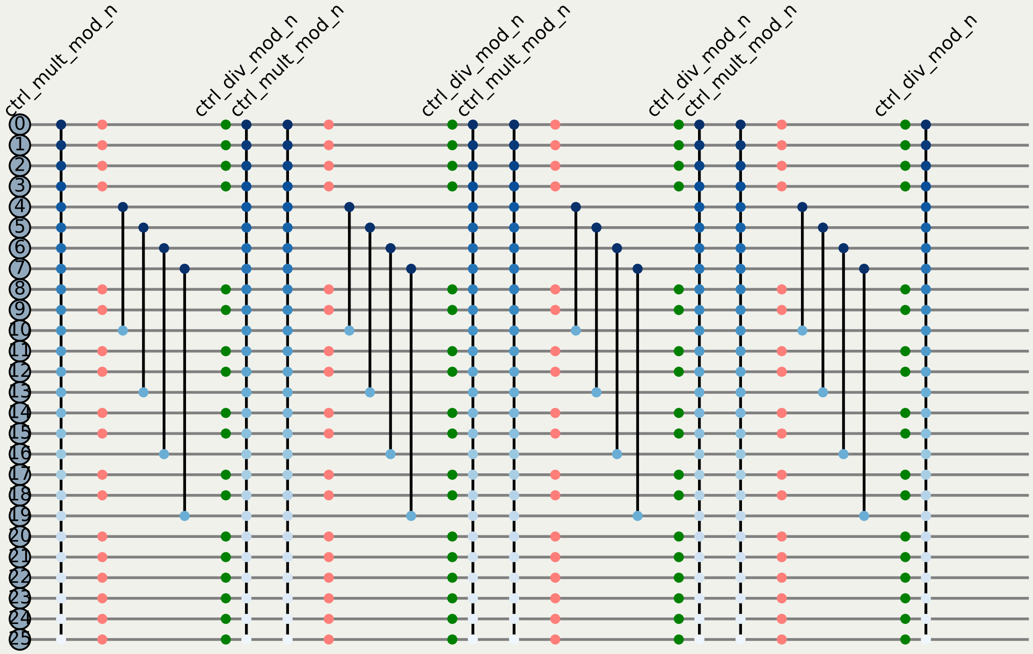
```
addition = carry_layer(1) + cnot_sum_pivot(1) + carry_sum_layer(1)
subtraction = carry_sum_layer(-1) + cnot_sum_pivot(-1) + carry_layer(-1)

plot_circuit(Qinit(11) + addition)
plot_circuit(Qinit(11) + subtraction)
plot_circuit(Qinit(13) + addition)
plot_circuit(Qinit(13) + subtraction)
```



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

Shor circuit

```python
exp_mod_n = tuple()
for k in range(n):
  exp_mod_n += (
    Qpivot(mapping=ctrl_mult(a ** (2**k), N, n, ctrl=k, divide=False))
    + swap_bx
    + Qpivot(mapping=ctrl_mult(a ** (2**k), N, n, ctrl=k, divide=True))
  )

hierq = Qinit(nq, tensors=tensors) + exp_mod_n
```

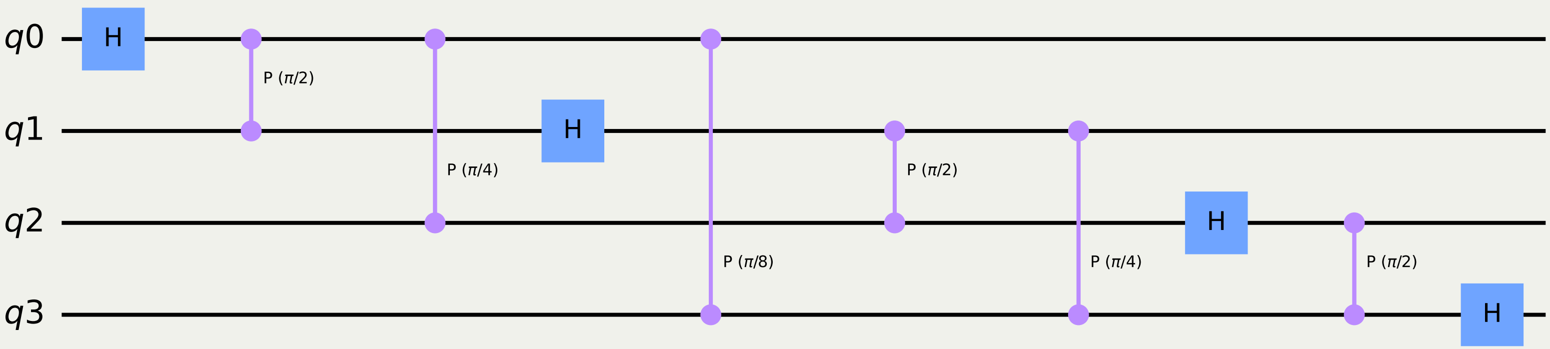


https://github.com/matt-lourens/hierarqcal

Quantum Fourier Transform

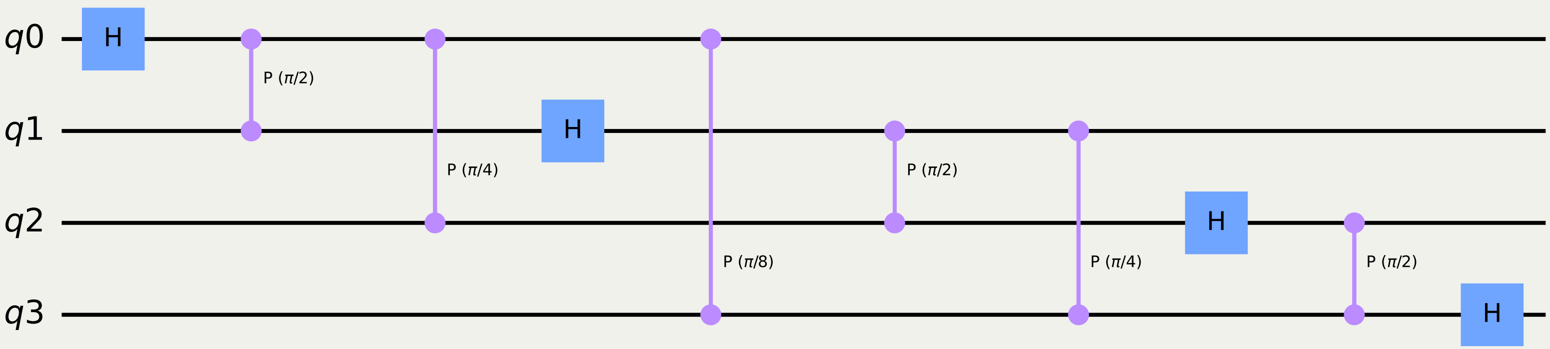
Quantum Fourier Transform

```
# QFT circuit
n = 4
qft = (
  Qpivot(mapping=Qunitary("h()^0"))
  + Qpivot(
    mapping=Qunitary("cp(x)^01"),
    share_weights=False,
    symbol_fn=lambda x, ns, ne: np.pi * 2 ** (-ne),
  )
  + Qmask("1*")
) * n
qft_n = Qinit(n) + qft

circuit_qft = qft_n(backend="qiskit", barriers=False)
circuit_qft.draw("mpl")
```



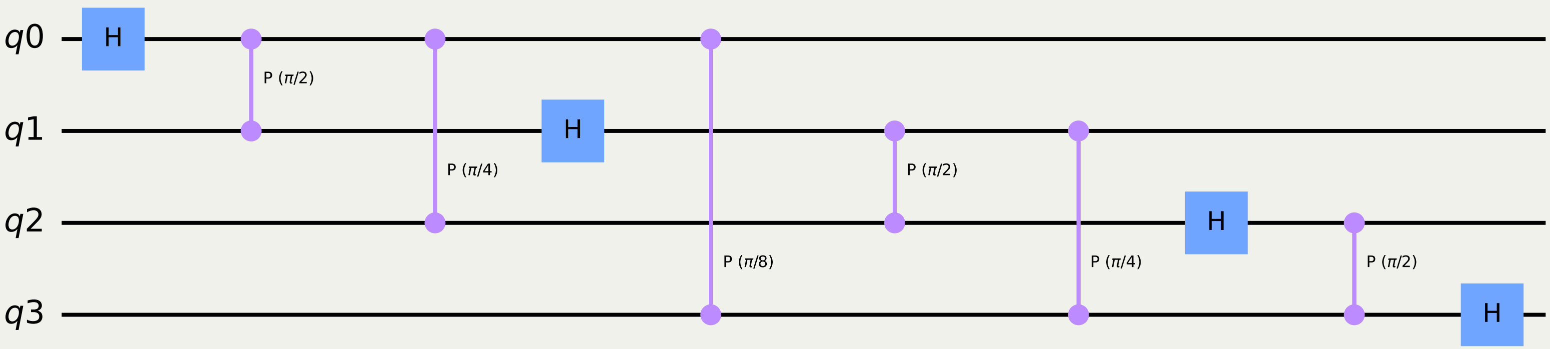
Quantum Fourier Transform

```python
# QFT circuit
n = 4
qft = (
  Qpivot(mapping=Qunitary("h()^0"))
  + Qpivot(
    mapping=Qunitary("cp(x)^01"),
    share_weights=False,
    symbol_fn=lambda x, ns, ne: np.pi * 2 ** (-ne),
  )
  + Qmask("1*")
) * n
qft_n = Qinit(n) + qft

circuit_qft = qft_n(backend="qiskit", barriers=False)
circuit_qft.draw("mpl")
```



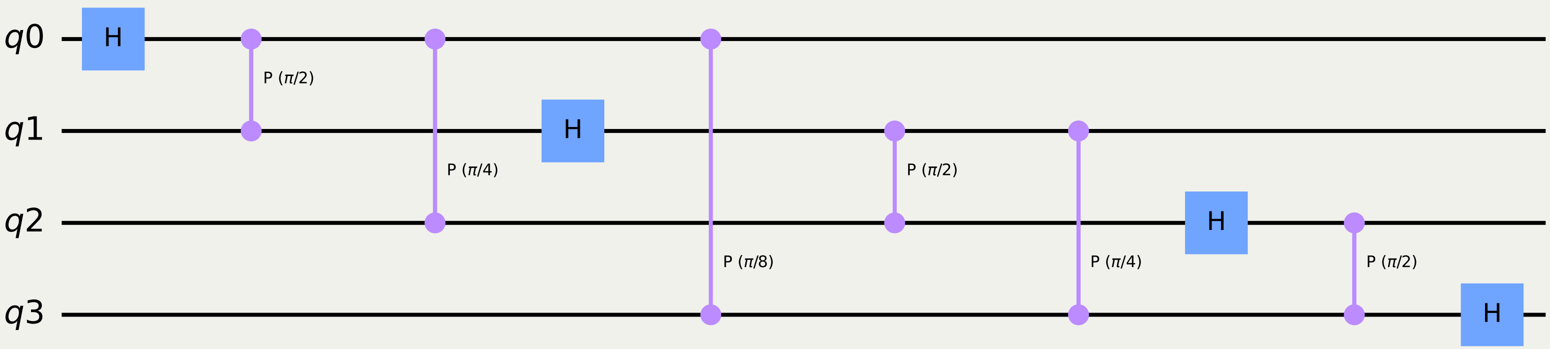
Quantum Fourier Transform

```python
# QFT circuit
n = 4
qft = (
  Qpivot(mapping=Qunitary("h()^0"))
  + Qpivot(
    mapping=Qunitary("cp(x)^01"),
    share_weights=False,
    symbol_fn=lambda x, ns, ne: np.pi * 2 ** (-ne),
  )
  + Qmask("1*")
) * n
qft_n = Qinit(n) + qft

circuit_qft = qft_n(backend="qiskit", barriers=False)
circuit_qft.draw("mpl")
```



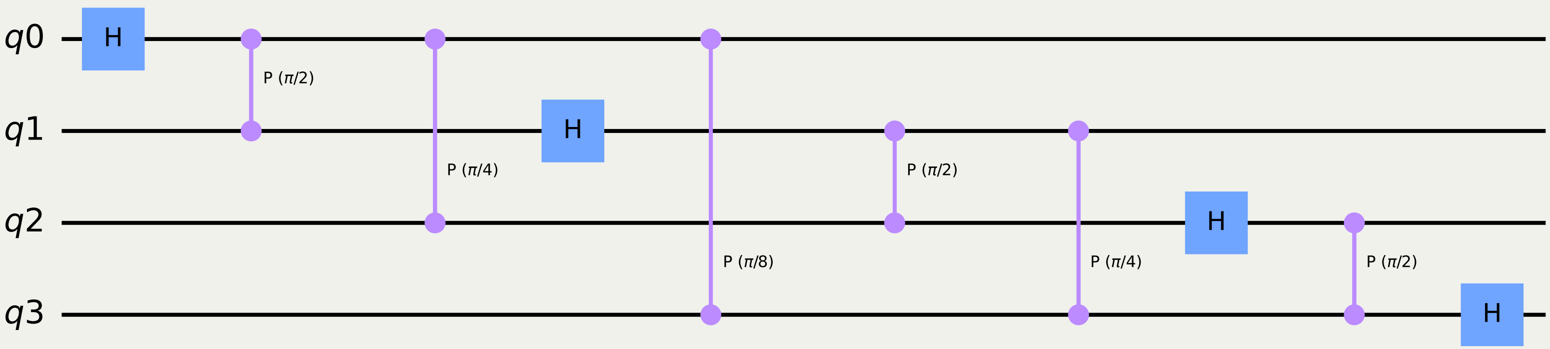
Quantum Fourier Transform

```python
# QFT circuit
n = 4
qft = (
  Qpivot(mapping=Qunitary("h()^0"))
  + Qpivot(
    mapping=Qunitary("cp(x)^01"),
    share_weights=False,
    symbol_fn=lambda x, ns, ne: np.pi * 2 ** (-ne),
  )
  + Qmask("1*")
) * n
qft_n = Qinit(n) + qft

circuit_qft = qft_n(backend="qiskit", barriers=False)
circuit_qft.draw("mpl")
```



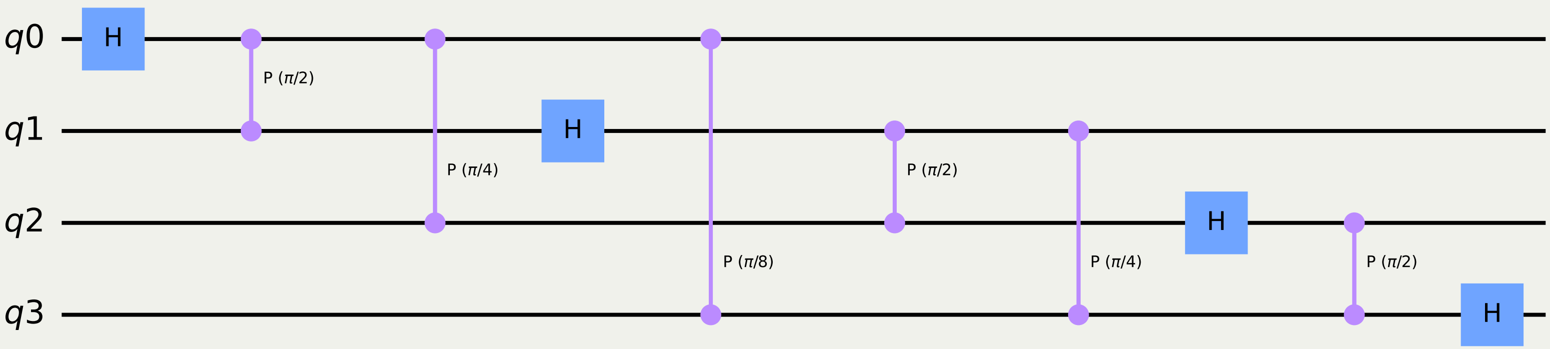
Quantum Fourier Transform

```python
# QFT circuit
n = 4
qft = (
  Qpivot(mapping=Qunitary("h()^0"))
  + Qpivot(
    mapping=Qunitary("cp(x)^01"),
    share_weights=False,
    symbol_fn=lambda x, ns, ne: np.pi * 2 ** (-ne),
  )
  + Qmask("1*")
) * n
qft_n = Qinit(n) + qft

circuit_qft = qft_n(backend="qiskit", barriers=False)
circuit_qft.draw("mpl")
```



Quantum Fourier Transform

```python
# QFT circuit
n = 4
qft = (
  Qpivot(mapping=Qunitary("h()^0"))
  + Qpivot(
    mapping=Qunitary("cp(x)^01"),
    share_weights=False,
    symbol_fn=lambda x, ns, ne: np.pi * 2 ** (-ne),
  )
  + Qmask("1*")
) * n
qft_n = Qinit(n) + qft

circuit_qft = qft_n(backend="qiskit", barriers=False)
circuit_qft.draw("mpl")
```



Classical Adder

Classical Adder

```python
def half_adder(bits, symbols=None, state=None):
  b1, b2 = state[bits[0]], state[bits[1]]
  xor = b1 ^ b2
  carry = b1 and b2
  state[bits[0]] = carry
  state[bits[1]] = xor
  return state

def or_top(bits, symbols=None, state=None):
  b1, b2 = state[bits[0]], state[bits[1]]
  state[bits[0]] = b1 or b2
  return state
```

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

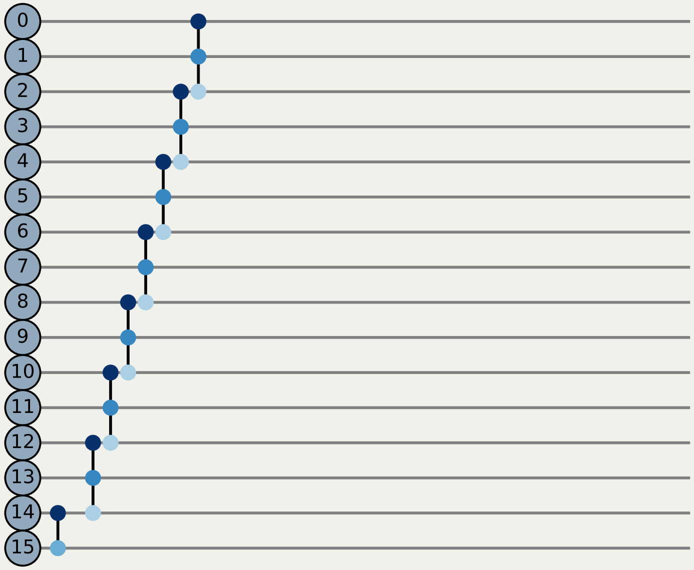
Classical Adder

```python
def half_adder(bits, symbols=None, state=None):
  b1, b2 = state[bits[0]], state[bits[1]]
  xor = b1 ^ b2
  carry = b1 and b2
  state[bits[0]] = carry
  state[bits[1]] = xor
  return state

def or_top(bits, symbols=None, state=None):
  b1, b2 = state[bits[0]], state[bits[1]]
  state[bits[0]] = b1 or b2
  return state
```

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

Classical Adder

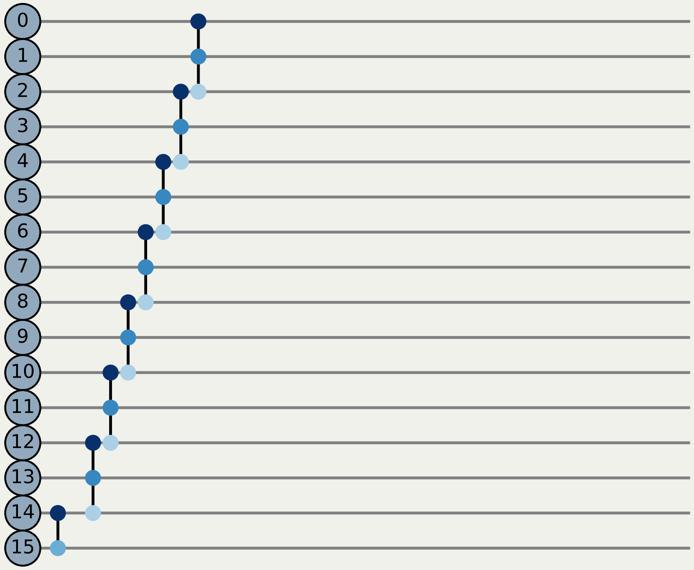
```python
# program
full_adder = (
  Qinit(3)
  + Qcycle(mapping=half_adder, boundary="open")
  + Qpivot(global_pattern="1*", merge_within="11"
        ,mapping=or_top)
)
addition = (
  Qinit(n)
  + Qpivot("*1", "11", mapping=half_adder)
  + Qcycle(step=2,
       edge_order=[-1],
       mapping=full_adder,
       boundary="open")
)
```



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

Classical Adder

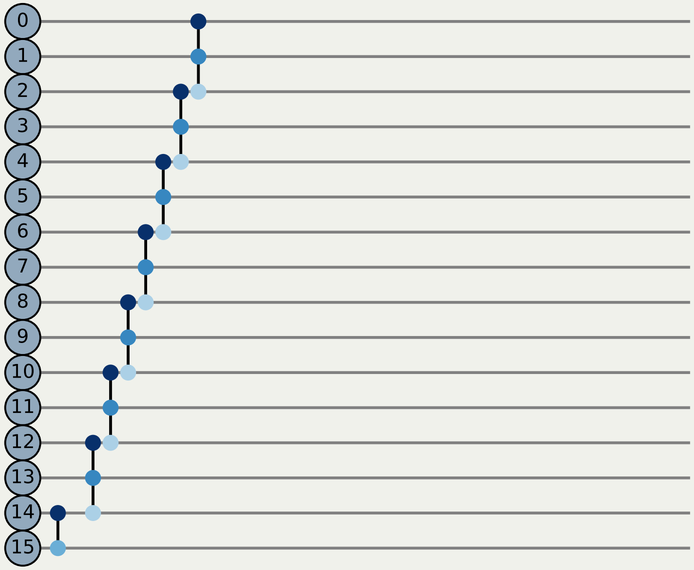
```
# program
full_adder = (
  Qinit(3)
  + Qcycle(mapping=half_adder, boundary="open")
  + Qpivot(global_pattern="1*", merge_within="11"
        ,mapping=or_top)
)
addition = (
  Qinit(n)
  + Qpivot("*1", "11", mapping=half_adder)
  + Qcycle(step=2,
       edge_order=[-1],
       mapping=full_adder,
       boundary="open")
)
```



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

Classical Adder

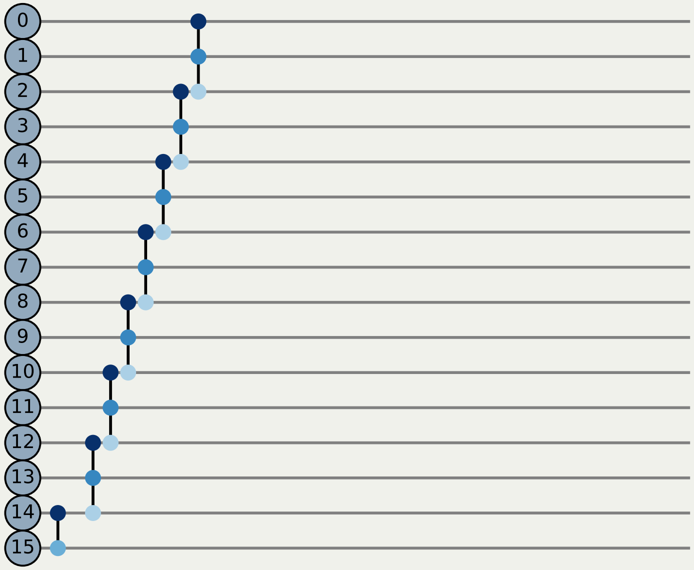
```
# program
full_adder = (
  Qinit(3)
  + Qcycle(mapping=half_adder, boundary="open")
  + Qpivot(global_pattern="1*", merge_within="11"
        ,mapping=or_top)
)
addition = (
  Qinit(n)
  + Qpivot("*1", "11", mapping=half_adder)
  + Qcycle(step=2,
        edge_order=[-1],
        mapping=full_adder,
        boundary="open")
)
```



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

Classical Adder

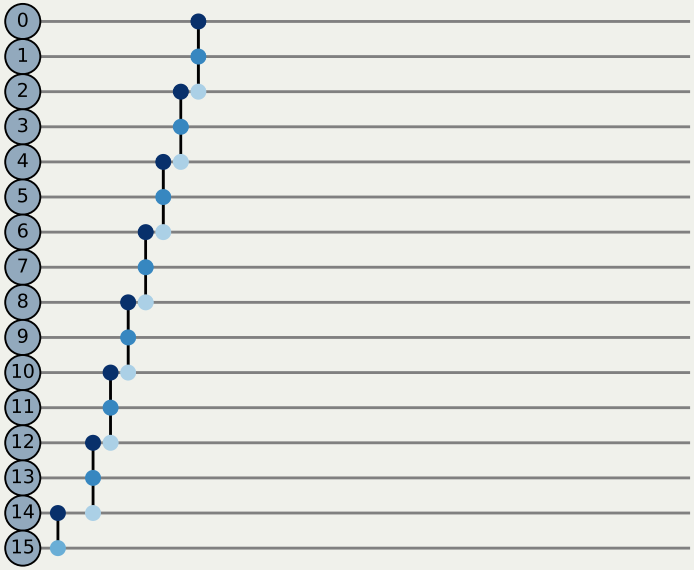
```
# program
full_adder = (
  Qinit(3)
  + Qcycle(mapping=half_adder, boundary="open")
  + Qpivot(global_pattern="1*", merge_within="11"
        ,mapping=or_top)
)
addition = (
  Qinit(n)
  + Qpivot("*1", "11", mapping=half_adder)
  + Qcycle(step=2,
       edge_order=[-1],
       mapping=full_adder,
       boundary="open")
)
```



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

Classical Adder

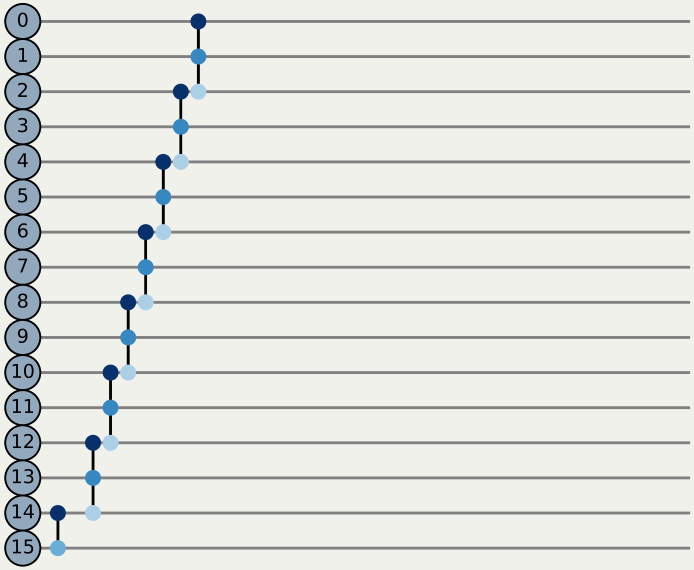
```
# program
full_adder = (
  Qinit(3)
  + Qcycle(mapping=half_adder, boundary="open")
  + Qpivot(global_pattern="1*", merge_within="11"
        ,mapping=or_top)
)
addition = (
  Qinit(n)
  + Qpivot("*1", "11", mapping=half_adder)
  + Qcycle(step=2,
       edge_order=[-1],
       mapping=full_adder,
       boundary="open")
)
```



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

Classical Adder

```
# program
full_adder = (
  Qinit(3)
  + Qcycle(mapping=half_adder, boundary="open")
  + Qpivot(global_pattern="1*", merge_within="11"
        ,mapping=or_top)
)
addition = (
  Qinit(n)
  + Qpivot("*1", "11", mapping=half_adder)
  + Qcycle(step=2,
       edge_order=[-1],
       mapping=full_adder,
       boundary="open")
)
```



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