



Perceval & Merlin Workshop

Agenda



Samuel HORSCH
Quantum Application Architect
at Quandela

samuel.horsch@quandela.com

❖ Quandela's QPU

❖ Building Photonic
Circuits with **Perceval**

❖ PyTorch Crash Course

❖ Building Hybrid
Quantum AI workflows
with **MerLin**



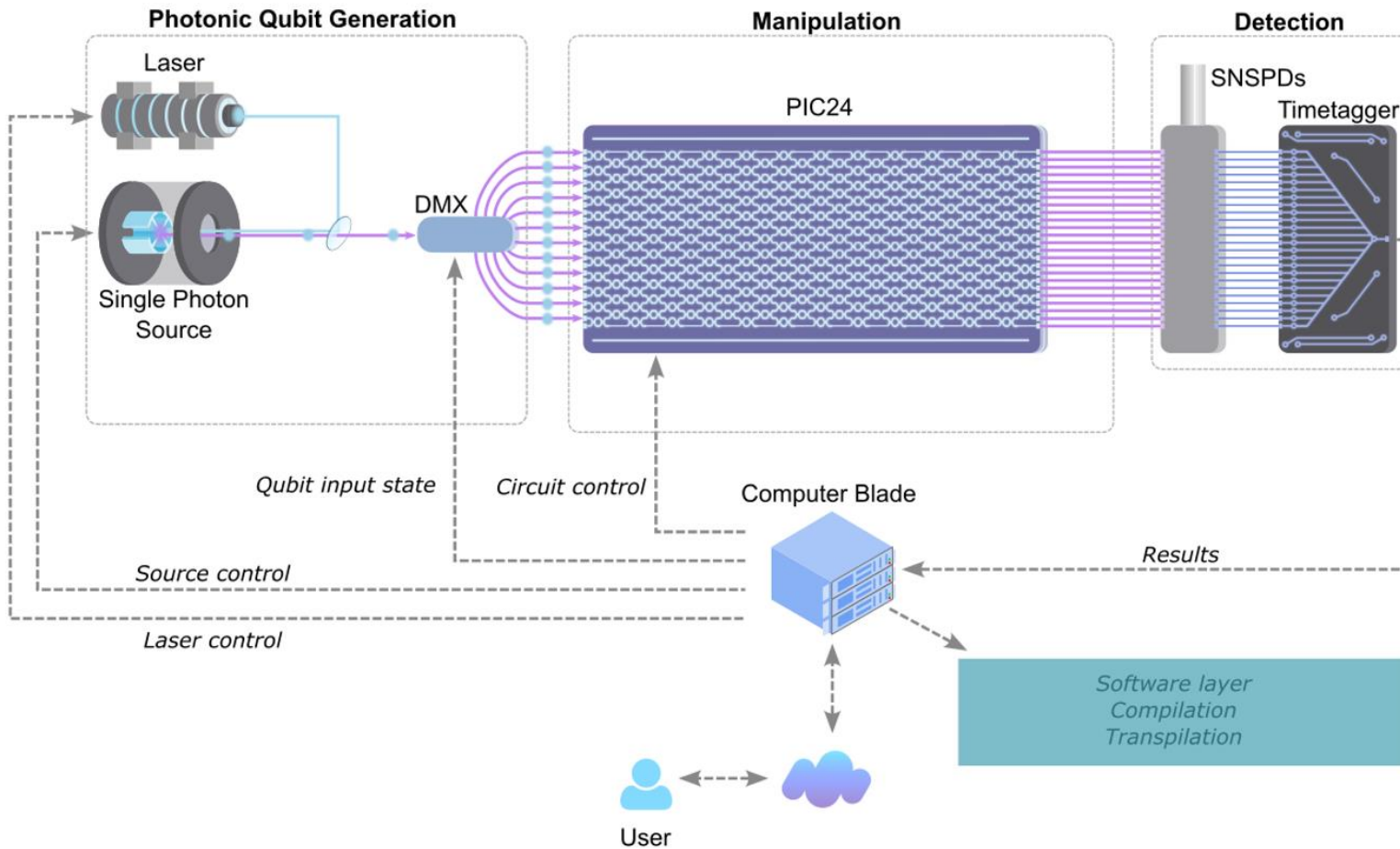
Download Interactive Notebook
to follow along the Workshop

01.

Introduction to Quandelas QPU



Photonic QPU



02.

Introduction to Perceval



Overview

- Perceval was developed to enable:
 - Linear optical simulation.
 - Interface with QPUs on the cloud.
 - Accelerate photonic algorithm development
- Perceval works a lot like Qiskit. We can define manipulate quantum states, define quantum circuits, add noise and run with exact probabilities or with shots.
- We can run these circuits on Processors (either local or remote).
- We can compile these circuits into PyTorch using Merlin.



Overview

- Perceval requires Python 3.9 or higher.

- To create a new python environment.

```
python -m venv perceval_env
```

- Activate the python environment:

```
source perceval_env/bin/activate    macOS / Linux  
perceval_env\Scripts\activate      windows
```

- Install perceval

```
pip install perceval-quandela
```

- See the docs <https://perceval.quandela.net/docs/>

- See the GitHub: <https://github.com/quandela/perceval>

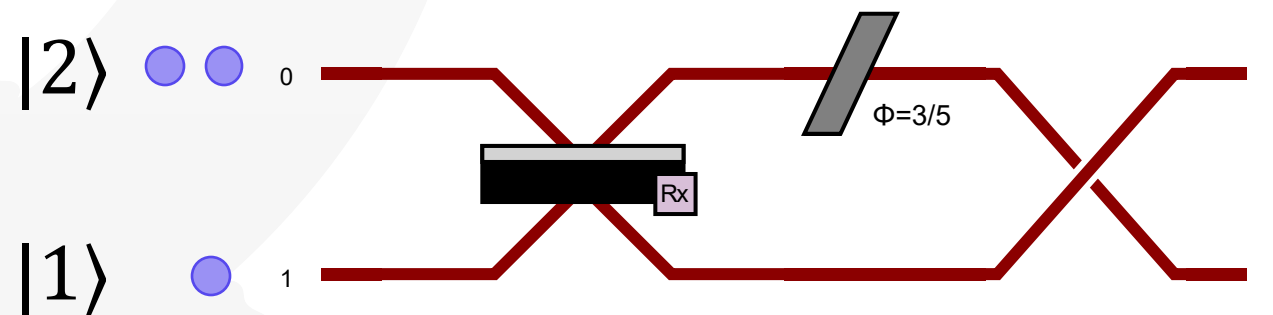
- To import classes from perceval:

```
import perceval as pcvl
```



Linear Optics

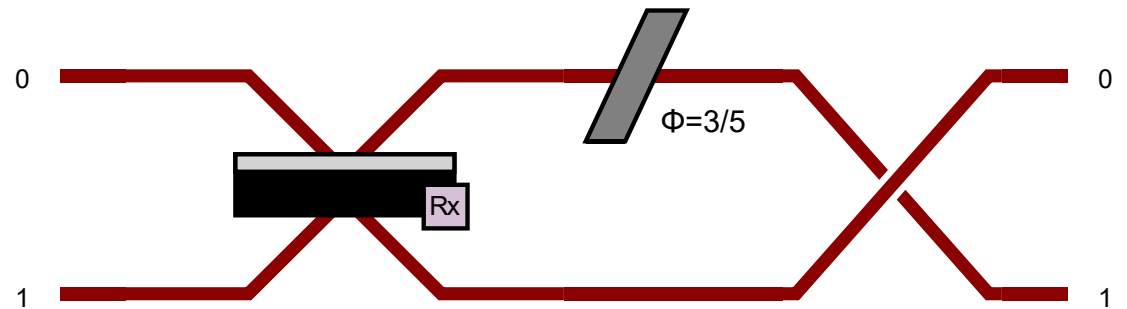
- At Quandela, we deal with photons in spatial modes.
- This is a **different** formalism than qubit-based quantum computing.
- We can set up a circuit with phase shifters (add phase to photons) and beamsplitters – split photons into superpositions.
- We look at the trajectory of some initial state of photons $|2, 1\rangle$ across some linear optical circuit.



Linear optical Circuits

- We can create beamsplitter, phase shifter gates and permutation gates easily & display the circuit.

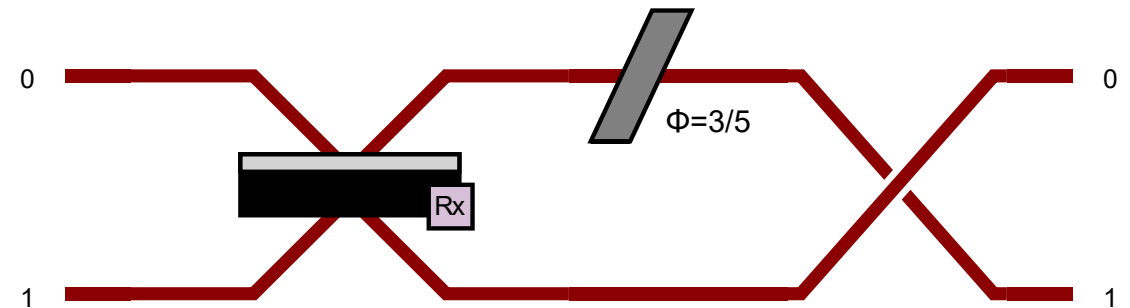
```
# Specify the number of modes.  
circuit = pcvl.Circuit(2)  
  
# Add a beamsplitter & phase shifter  
circuit.add(0, pcvl.BS())  
circuit.add(0, pcvl.PS(0.6))  
  
# Add permutation gate  
circuit.add(0, pcvl.PERM([1, 0]))  
  
pdisplay(circuit)
```



Try creating your own Circuit!

- We can create beamsplitter, phase shifter gates and permutation gates easily & display the circuit.

```
# Specify the number of modes.  
circuit = pcvl.Circuit(2)  
  
# Add a beamsplitter & phase shifter  
circuit.add(0, pcvl.BS())  
circuit.add(0, pcvl.PS(0.6))  
  
# Add permutation gate  
circuit.add(0, pcvl.PERM([1, 0]))  
  
pdisplay(circuit)
```



With `pcvl.P("theta")` you can create a Parameter instead of defining the value directly.

03.

Introduction to Merlin

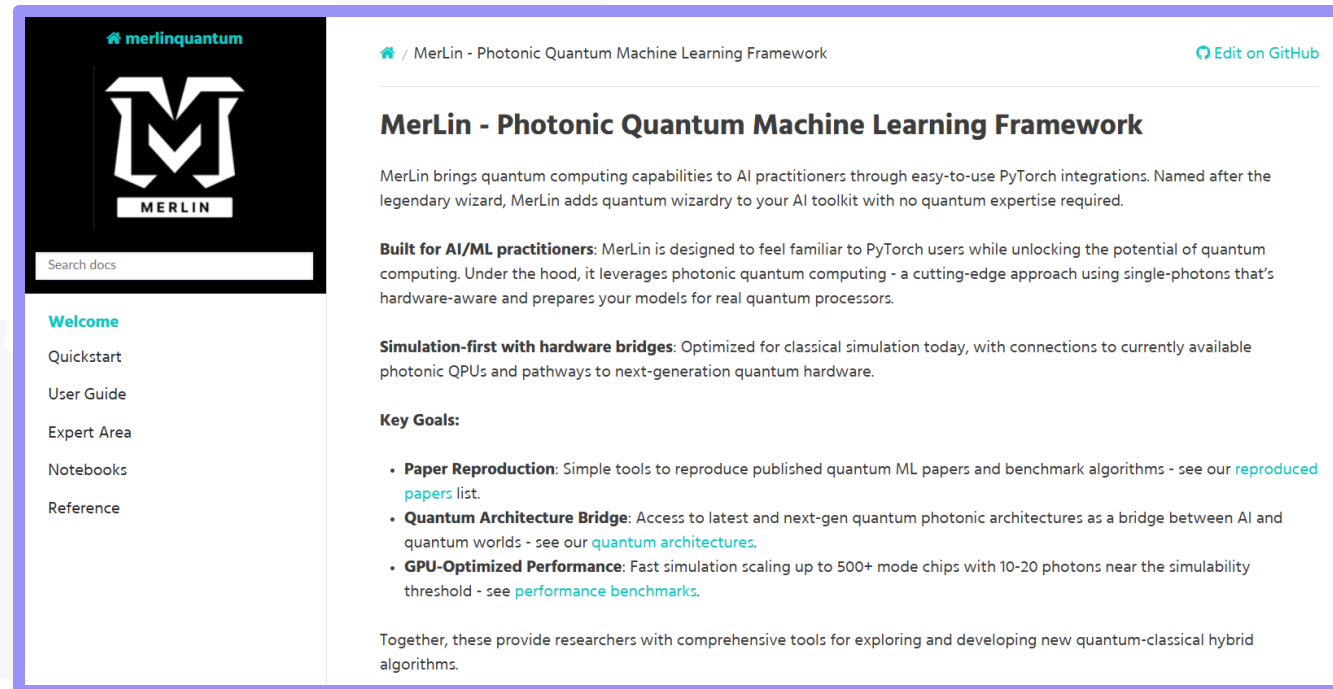


MerLin

- Install MerLin

```
pip install merlinquantum
```

- See the documentation: <https://merlinquantum.ai/>



merlinquantum

M
MERLIN

Search docs

Welcome

Quickstart

User Guide

Expert Area

Notebooks

Reference

Home / MerLin - Photonic Quantum Machine Learning Framework [Edit on GitHub](#)

MerLin - Photonic Quantum Machine Learning Framework

MerLin brings quantum computing capabilities to AI practitioners through easy-to-use PyTorch integrations. Named after the legendary wizard, MerLin adds quantum wizardry to your AI toolkit with no quantum expertise required.

Built for AI/ML practitioners: MerLin is designed to feel familiar to PyTorch users while unlocking the potential of quantum computing. Under the hood, it leverages photonic quantum computing - a cutting-edge approach using single-photons that's hardware-aware and prepares your models for real quantum processors.

Simulation-first with hardware bridges: Optimized for classical simulation today, with connections to currently available photonic QPUs and pathways to next-generation quantum hardware.

Key Goals:

- **Paper Reproduction:** Simple tools to reproduce published quantum ML papers and benchmark algorithms - see our [reproduced papers](#) list.
- **Quantum Architecture Bridge:** Access to latest and next-gen quantum photonic architectures as a bridge between AI and quantum worlds - see our [quantum architectures](#).
- **GPU-Optimized Performance:** Fast simulation scaling up to 500+ mode chips with 10-20 photons near the simulability threshold - see [performance benchmarks](#).

Together, these provide researchers with comprehensive tools for exploring and developing new quantum-classical hybrid algorithms.



PyTorch Crashcourse

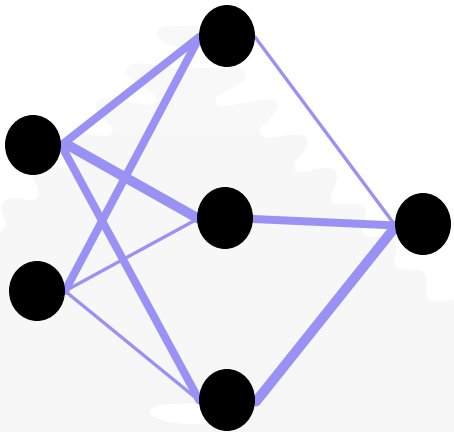
- PyTorch is a ML Python framework.
- The framework contains many of the same operations as numpy.

`numpy.array` → `torch.tensor (FloatTensor or LongTensor)`



PyTorch Crashcourse - Layers

Say we want to have a neural network consisting of two linear layers:



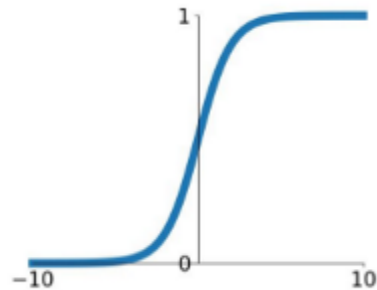
```
# Specify the number of nodes.  
linear1 = torch.nn.Linear(  
    in_features=2,  
    out_features=3  
)  
linear2 = torch.nn.Linear(  
    in_features=3,  
    out_features=1  
)  
X = torch.FloatTensor([[1.0, 2.0], [3.0, 4.0], [5.0, 6.0]])  
  
output = linear1(X)  
output = linear2(output)
```

PyTorch Crashcourse - Layers

We add non-linear activation functions to make the neural network more expressive between linear layers:

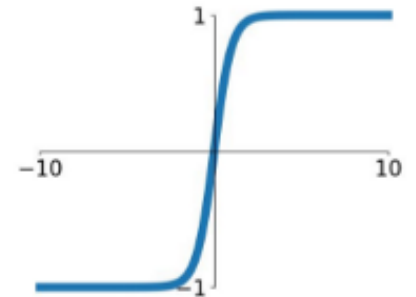
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



```
sigmoid_layer = torch.nn.Sigmoid()
```

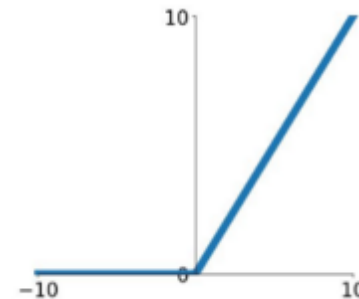
tanh

$$\tanh(x)$$


```
tanh_layer = torch.nn.Tanh()
```

ReLU

$$\max(0, x)$$



```
relu = torch.nn.ReLU()
```

Creating Neural Nets

```
class NeuralNetwork(torch.nn.Module):
    def __init__(self):
        super().__init__()

        self.linear1 = torch.nn.Linear(2, 3) # Register params here.
        self.relu = torch.nn.ReLU()
        self.linear2 = torch.nn.Linear(3, 1)

        self.network = torch.nn.Sequential(
            self.linear1, self.relu, self.linear2
        )

    def forward(self, X):
        return self.network(X)
```



PyTorch Crashcourse - Training

After specifying a neural network, we define the loss function and optimizer.

Training is done in a for loop like so:

```
optimizer = torch.optim.Adam(network.parameters(), lr=1e-2)

loss_function = torch.nn.CrossEntropyLoss()

for epoch in range(num_epochs):
    optimizer.zero_grad() # Reset the gradient.

    output = network(X)
    loss = loss_function(output, targets)

    loss.backward() # Calculate gradient
    optimizer.step() # Increment parameters
```

Quantum Layers with MerLin

Often in photonic machine learning, we have a universal interferometer and encode the data & training params into phase shifters and collect the output as the probabilities of each output state.

In MerLin, this can be defined as a torch layer in one step:

```
quantum_layer = QuantumLayer.simple(  
    input_size=3,  
    n_params=50  
)  
  
X = torch.rand(10, 3)  
output = quantum_layer(X)
```

Try it yourself!

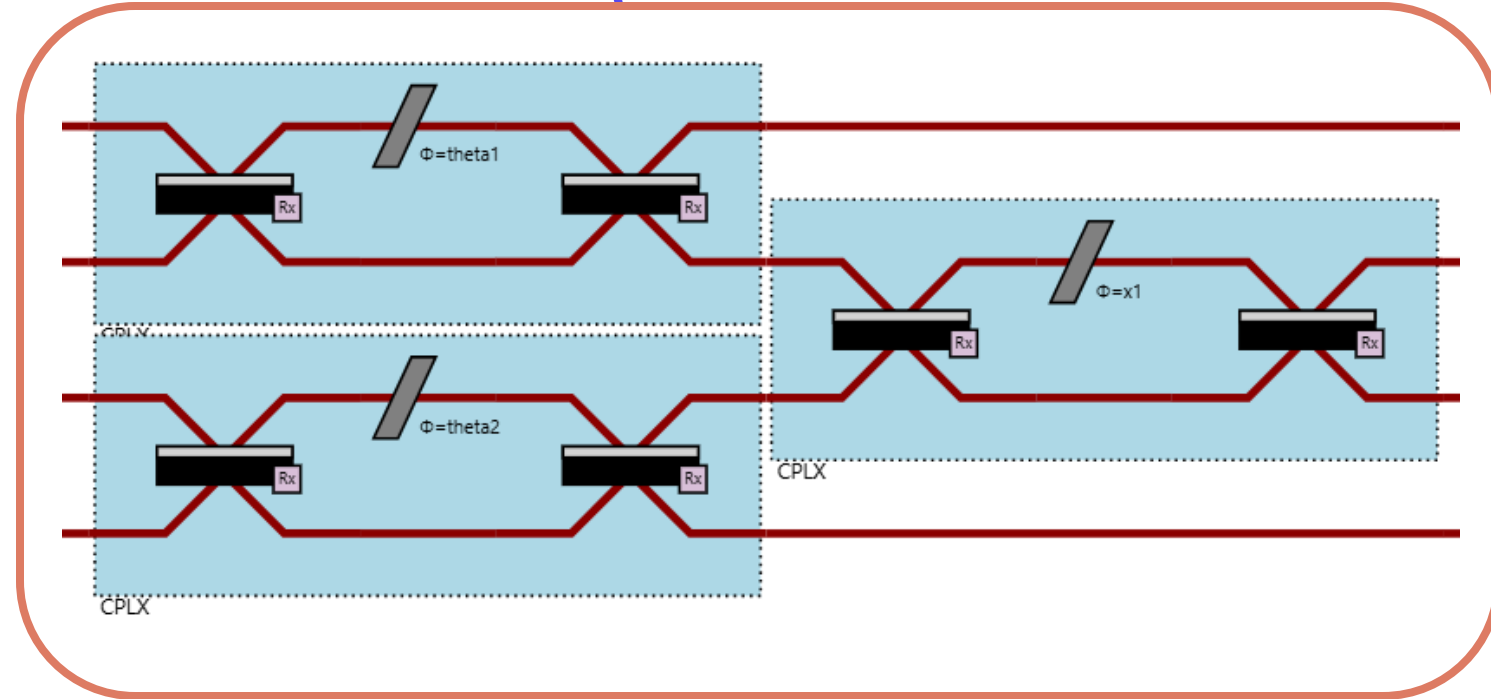
Often in photonic machine learning, we have a universal interferometer and encode the data & training params into phase shifters and collect the output as the probabilities of each output state.

In MerLin, this can be defined as a torch layer in one step:

```
quantum_layer = QuantumLayer.simple(  
    input_size=3,  
    n_params=50  
)  
  
X = torch.rand(10, 3)  
output = quantum_layer(X)
```

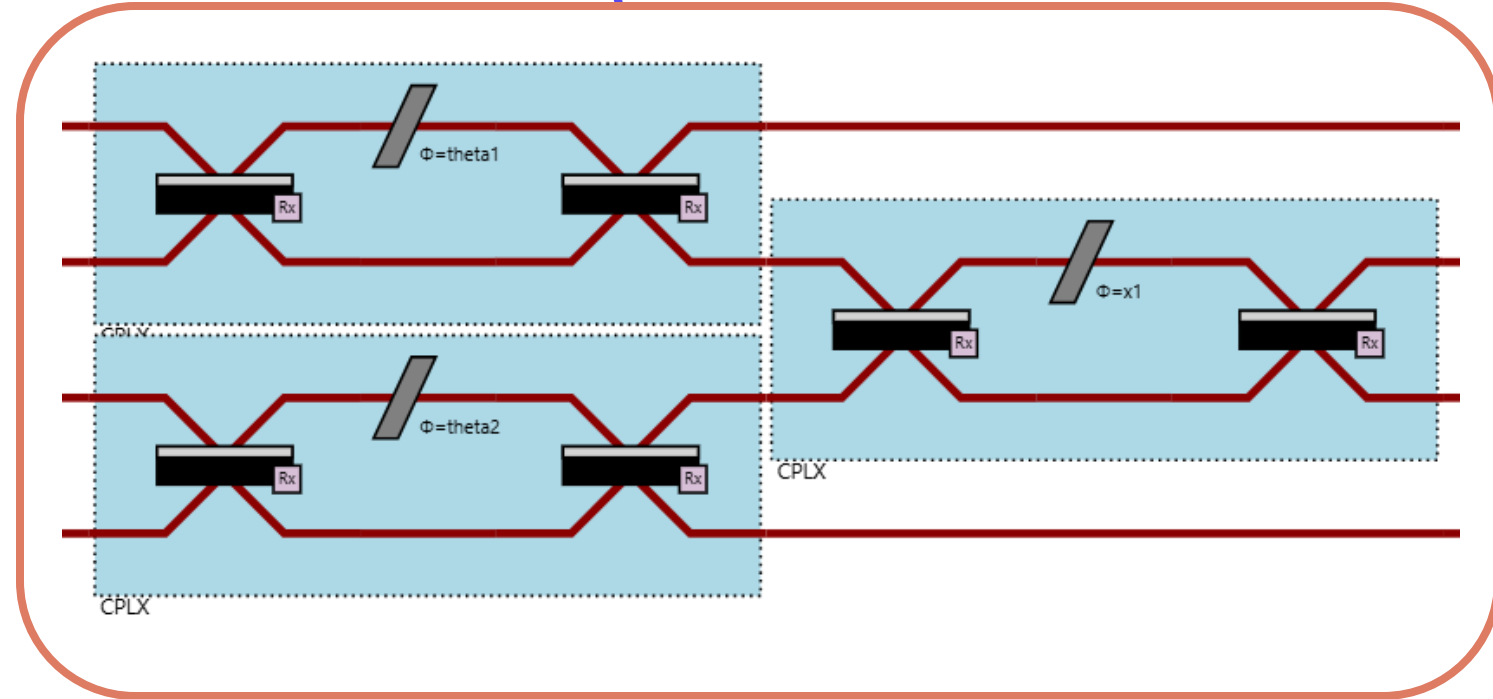
Circuit - Quantum Layers conversion

```
quantum_layer = QuantumLayer(  
    input_size=1,  
    output_size=3,  
    circuit=circuit,  
    no_bunching=False,  
    trainable_parameters=["theta"],  
    input_parameters=["x1"],  
    input_state=[1, 0, 1, 0],  
    output_mapping_strategy=OutputMappingStrategy.LINEAR,  
)
```



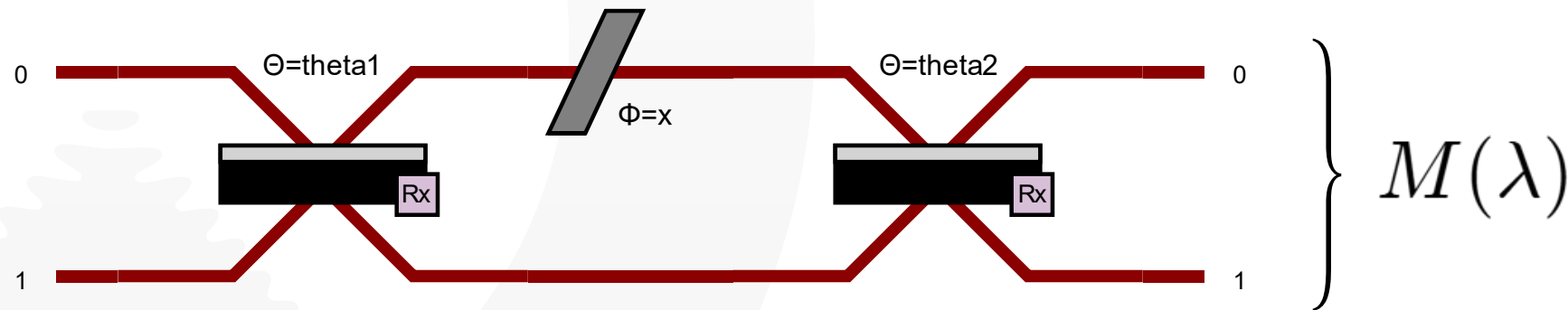
Try it yourself!

```
quantum_layer = QuantumLayer(  
    input_size=1,  
    output_size=3,  
    circuit=circuit,  
    no_bunching=False,  
    trainable_parameters=["theta"],  
    input_parameters=["x1"],  
    input_state=[1, 0, 1, 0],  
    output_mapping_strategy=OutputMappingStrategy.LINEAR,  
)
```



Final Challenge – Train a MZI

- Define a Perceval circuit which embeds data parameter, X in the middle phase-shifter and training parameters, θ_1 and θ_2 .



- Can you train the circuit parameters & measurement operator to approximate a Gaussian with $\sigma = 0.3$ with input state $|2, 0\rangle$.