Introduction to Quantum Machine learning

Qiyu Sha

IHEP

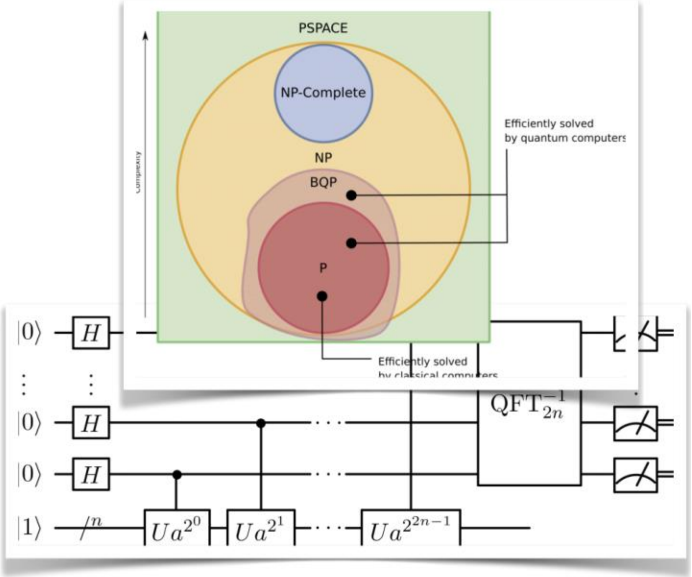
Quantum Computing and Machine Learning Workshop 2024

Overview

Qubit Type	Pros/Cons	Select Players
Superconducting	***Pros***: High gate speeds and fidelities. Can leverage standard lithographic processes. Among first qubit modalities so has a head start. ***Cons***: Requires cryogenic cooling; short coherence times; microwave interconnect frequencies still not well understood.	rigetti, Google, IBM Q, QuTech, OQC, IQM, qci Quantum Circuits, Inc, 本源量子 Origin Quantum
Trapped Ions	***Pros***: Extremely high gate fidelities and long coherence times. Extreme cryogenic cooling not required. Ions are perfect and consistent. ***Cons***: Slow gate times/ operations and low connectivity between qubits. Lasers hard to align and scale. Ultra-high vacuum required. Ion charges may restrict scalability.	IONQ, AQT, QUANTINUUM, oxford ionics, Universal Quantum
Photonics	***Pros***: Extremely fast gate speeds and promising fidelities. No cryogenics or vacuums required. Small overall footprint. Can leverage existing CMOS fabs. ***Cons***: Noise from photon loss; each program requires its own chip. Photons don't naturally interact so 2Q gate challenges.	PsiQuantum, XANADU, QUIX QUANTUM, ORCA Computing
Neutral Atoms	***Pros***: Long coherence times. Atoms are perfect and consistent. Strong connectivity, including more than 2Q. External cryogenics not required. ***Cons***: Requires ultra-high vacuums. Laser scaling challenging.	ColdQuanta, QuEra COMPUTING INC., atom computing, PASQAL
Silicon Spin/Quantum Dots	***Pros***: Leverages existing semiconductor technology. Strong gate fidelities and speeds. ***Cons***: Requires cryogenics. Only a few entangled gates to-date with low coherence times. Interference/cross-talk challenges.	intel, Silicon Quantum Computing, diraq, QUANTUM MOTION, QUANTUM BRILLIANCE

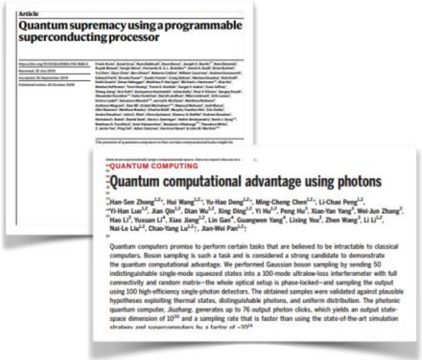
Computers

Algorithms

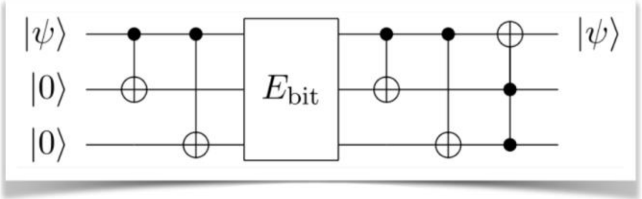


Programming

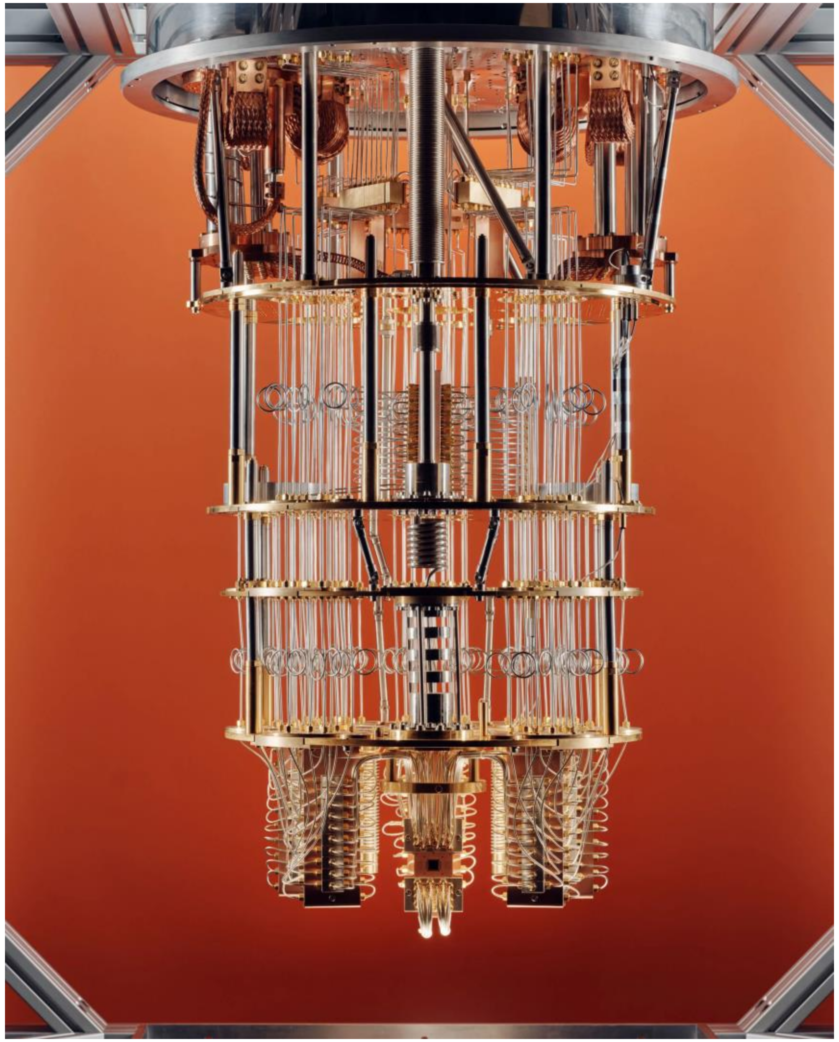
Advantage



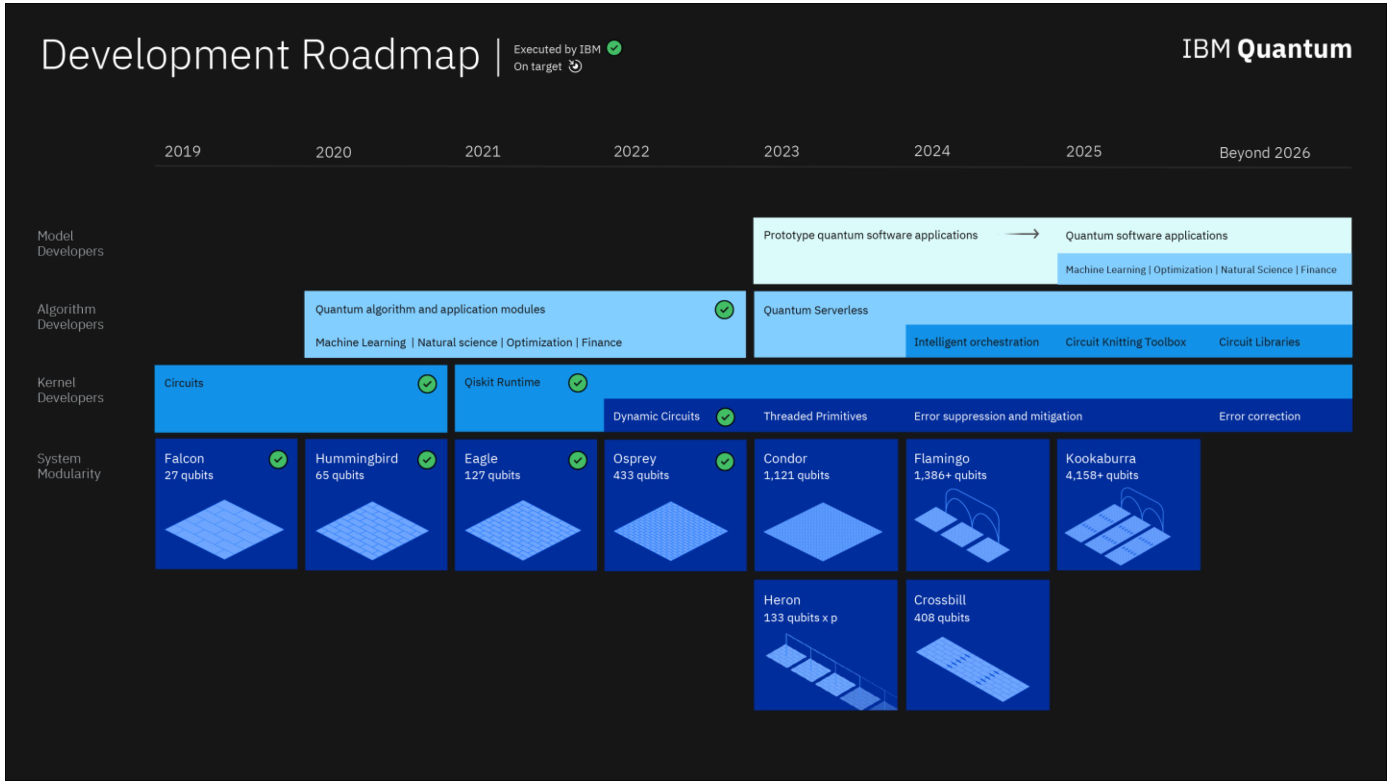
Error Correction



Introduction---IBM Quantum Computer



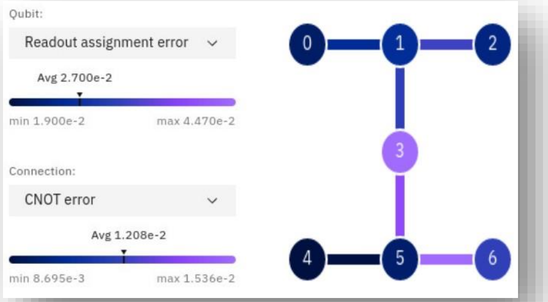
Credited to Thomas Prior for TIME



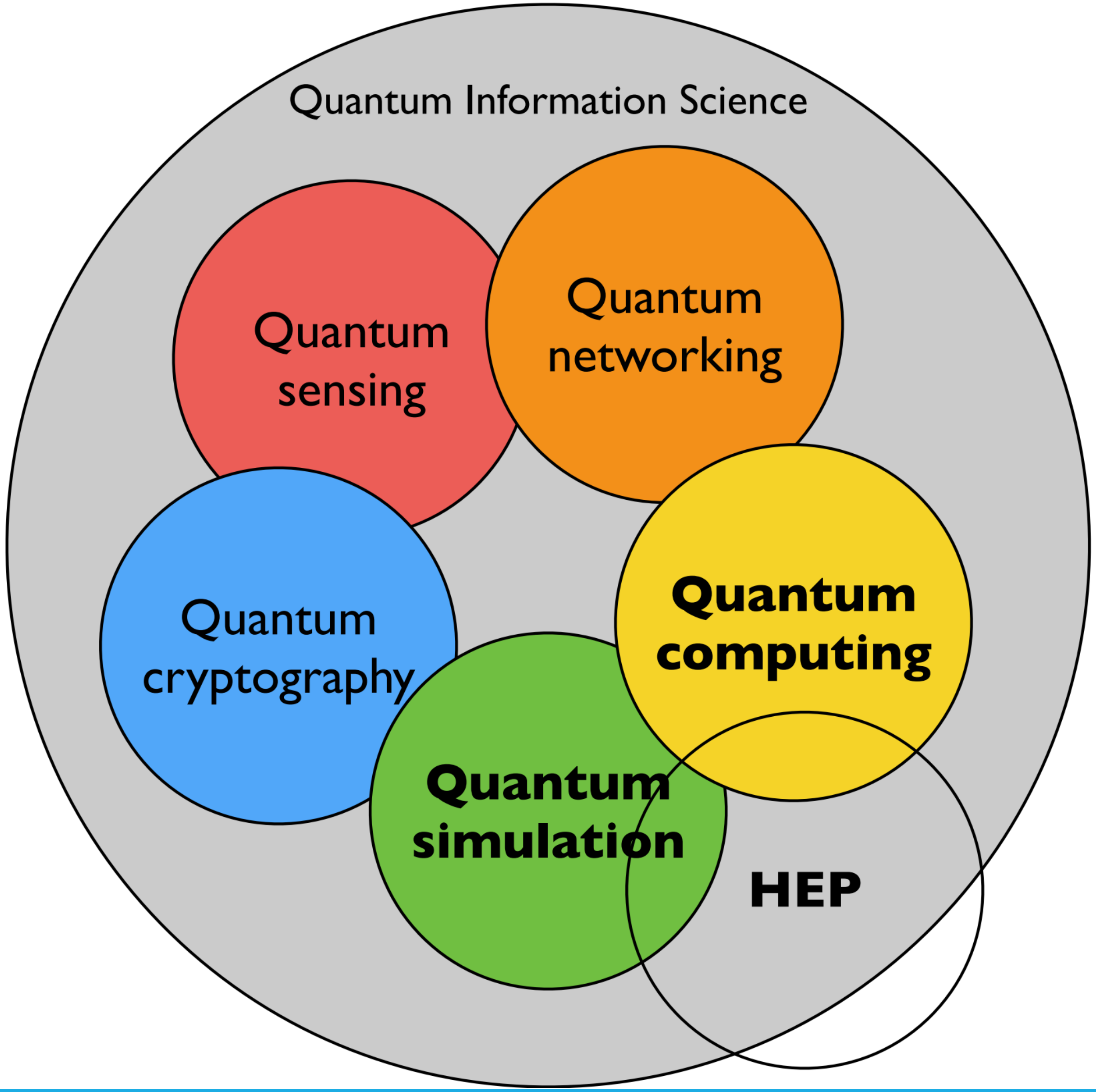
IBM has ambitious pursuits:

- 433-qubits IBM Quantum Osprey
- Three times larger than the Eagle processor (127-qubits)
- Going up to 10k-100k qubits.

Now, IBM provides up to 127 qubits for free.

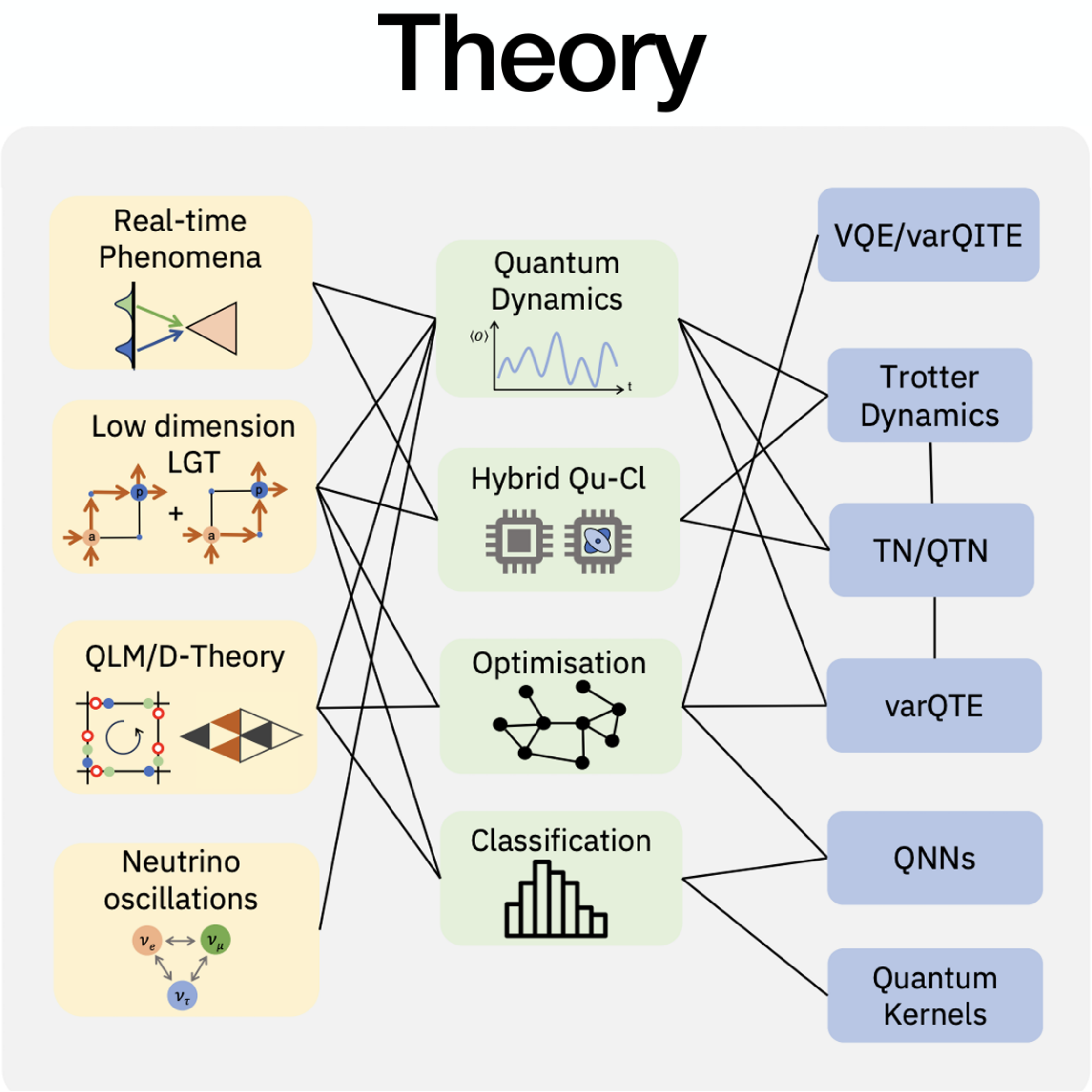


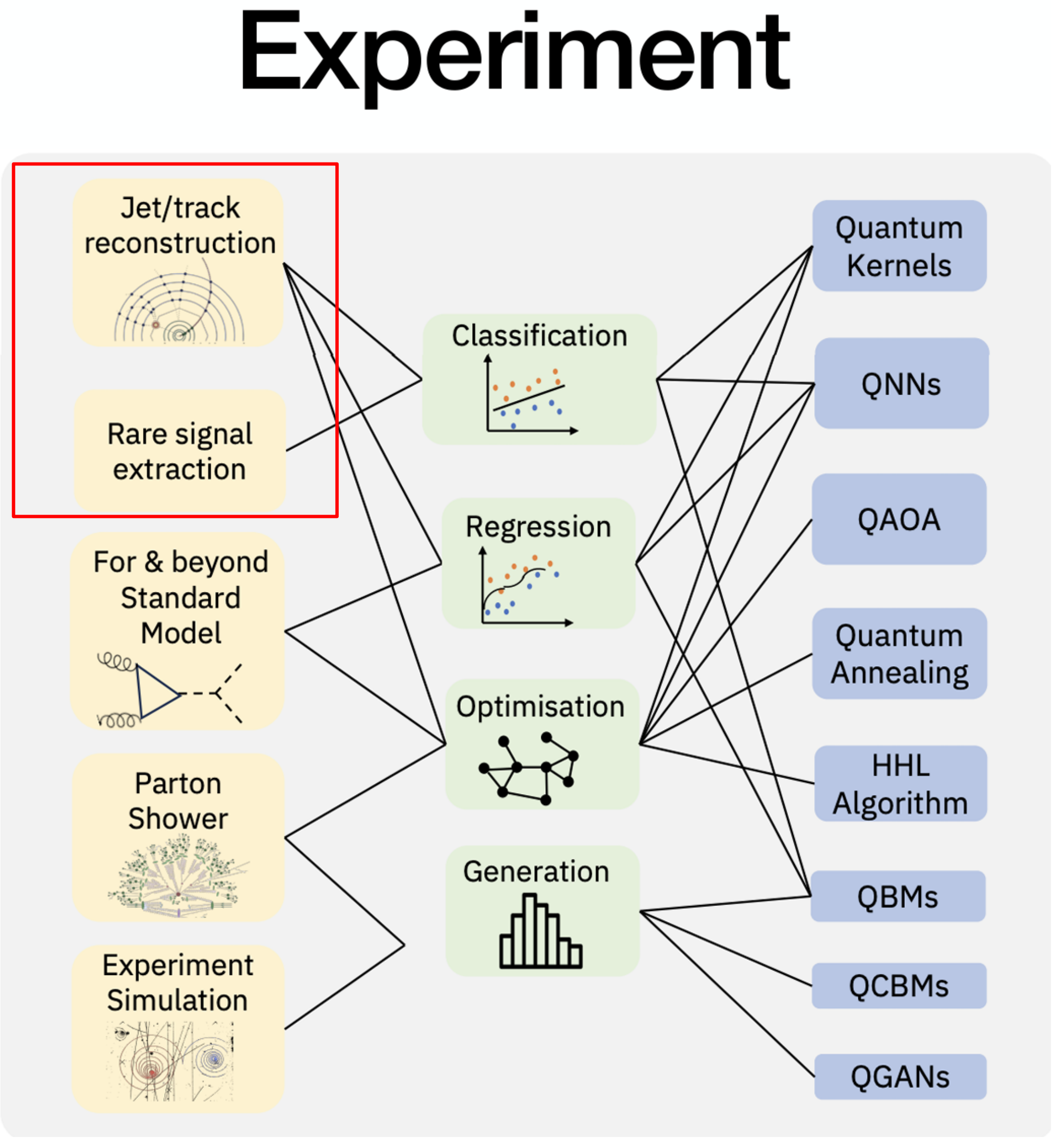
Overview



Quantum Information Science
Quantum sensing
Quantum networking
Quantum cryptography
Quantum computing
Quantum simulation
HEP

Outline of today





Quantum transformer

- Quantum Transformer:
  - At the core of any Transformer sits the so-called ***Multi-Headed Attention.***
  - We apply three different linear transformations $W_Q, W_K,$ and $W_V,$ to each element of the input sequence to transform each element embedding into some other internal representation states called Query (Q), Key (K) and Value (V). These states are then passed to the function that calculates the attention weights, which is simply defined as:

$$Attention(Q,K,V) = softmax_k\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

  - To promote the Transformer from the classical to quantum real, one can simply replace the linear transformations $W_Q, W_K,$ and W_V with variational quantum circuits.

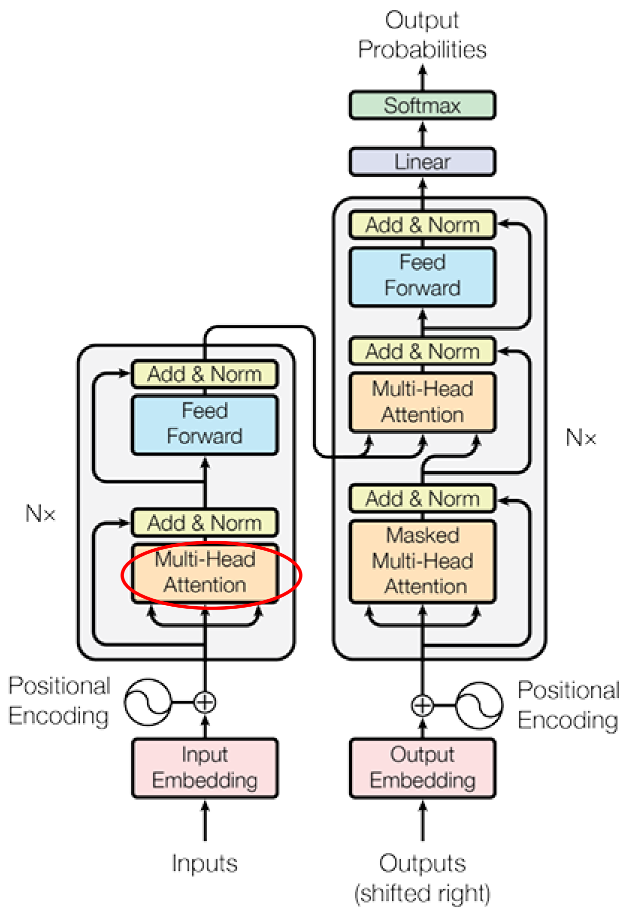


Figure 1: The Transformer - model architecture.

Code detail

Quantum transformer: (This code following here)

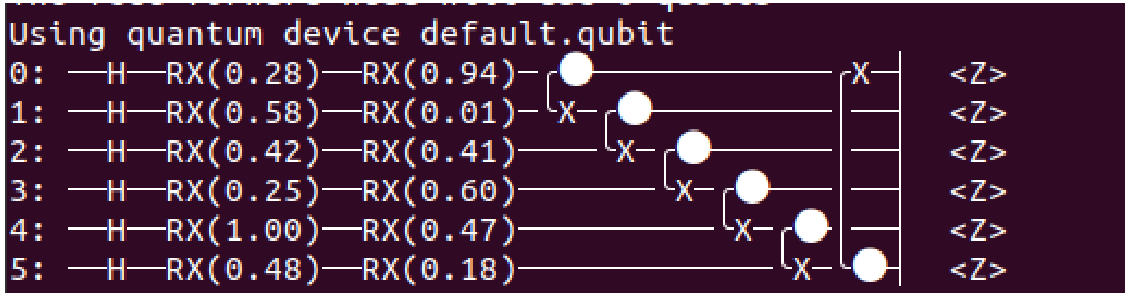
- The MultiHeadAttentionQuantum block
- Change the linear transformations.

```python
class MultiHeadAttentionClassical(MultiHeadAttentionBase):
    def __init__(self,
                embed_dim: int,
                num_heads: int,
                dropout=0.1,
                mask=None,
                use_bias=False):
        super(MultiHeadAttentionClassical, self).__init__(embed_dim=embed_dim, num_heads=num_heads, dropout=dropout, mask=mask, use_bias=use_bias)

        self.k_linear = nn.Linear(embed_dim, embed_dim, bias=use_bias)
        self.q_linear = nn.Linear(embed_dim, embed_dim, bias=use_bias)
        self.v_linear = nn.Linear(embed_dim, embed_dim, bias=use_bias)
        self.combine_heads = nn.Linear(embed_dim, embed_dim, bias=use_bias)
        self.head_dim = embed_dim // num_heads

    def forward(self, x, mask=None):
        batch_size, seq_len, embed_dim = x.size()
        assert embed_dim == self.embed_dim, f"Input embedding ({embed_dim}) does not match layer embedding size ({self.embed_dim})"

        K = self.k_linear(x)
        Q = self.q_linear(x)
        V = self.v_linear(x)

        x = self.downstream(Q, K, V, batch_size, mask)
        output = self.combine_heads(x)
        return output
```

```python
self.n_qubits = n_qubits
self.n_qlayers = n_qlayers
self.q_device = q_device
self.head_dim = embed_dim // num_heads
if 'qulacs' in q_device:
    self.dev = qml.device(q_device, wires=self.n_qubits, gpu=True)
elif 'braket' in q_device:
    self.dev = qml.device(q_device, wires=self.n_qubits, parallel=True)
else:
    self.dev = qml.device(q_device, wires=self.n_qubits)
def _circuit(inputs, weights):
    for i in range(n_qubits):
        qml.Hadamard(wires=i)
    qml.AngleEmbedding(inputs, wires=range(n_qubits))
    qml.BasicEntanglerLayers(weights, wires=range(n_qubits))
    return [qml.expval(qml.PauliZ(wires=i)) for i in range(n_qubits)]

self.qlayer = qml.QNode(_circuit, self.dev, interface="torch")
self.weight_shapes = {"weights": (n_qlayers, n_qubits)}

#draw plots
weights = np.random.random([n_qlayers, n_qubits])
X = np.random.rand(6)
print(qml.draw(self.qlayer, expansion_strategy="device")(X,weights))

print(f"weight_shapes = (n_qlayers, n_qubits) = ({n_qlayers}, {self.n_qubits})")

self.k_linear = qml.qnn.TorchLayer(self.qlayer, self.weight_shapes)
self.q_linear = qml.qnn.TorchLayer(self.qlayer, self.weight_shapes)
self.v_linear = qml.qnn.TorchLayer(self.qlayer, self.weight_shapes)
self.combine_heads = qml.qnn.TorchLayer(self.qlayer, self.weight_shapes)
```

Quantum transformer model

- We make use of Xanadu's PennyLane quantum machine learning library to add quantum layers.
  - Add a function to the class and performs the quantum calculation (with a circuit) .
  - Wrap this circuit with a "QNode" to tell TensorFlow how to calculate the gradient with the parameter-shift rule.



  - Finally, we create a KerasLayer to handle the I/O within the hybrid neural network.($W_Q, W_K,$ and W_V)
  - Other part is same as the classical transformer.
  - https://github.com/shaqiyu/Quantum_transformer

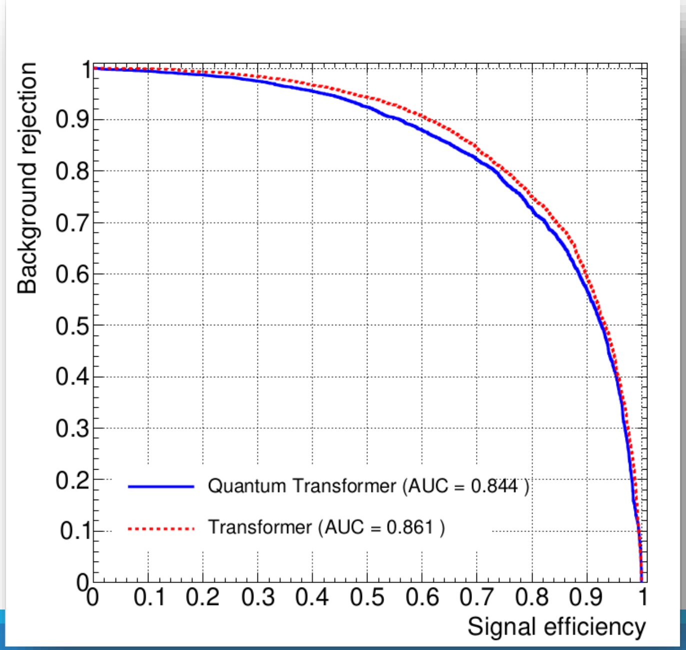
Quantum transformer

- The dataset we used now is the CEPC MC sample
  - $e^+e^- \to ZH \to q\bar{q}\gamma\gamma$ (signal) and $e^+e^- \to (Z/\gamma^*)\gamma\gamma$ (background)
- Simulator: Pennlylane default device.
  - Time consuming: O(n), ~80 mins in CPU for 10k dataset with one epoch and one block(Q_layer).
- Current results (Use CPU): ~76% acc on validation dataset both in Quantum transformer and classical transformer in 20k dataset (10k train, 10k val).
  - Quantum:

```
Epoch 8/30
Info in <TCanvas::Print>: pdf file ./plot/ROCs/ROC_10000_train.pdf has been created
Info in <TCanvas::Print>: pdf file ./plot/ROCs/ROC_10000_val.pdf has been created
Epoch: 08 | Epoch Time: 140m 18s
        Train Loss: 0.510 | Train Acc: 76.24%
         Val. Loss: 0.500 |  Val. Acc: 76.52%
```

  - Classical:

```
Epoch 10/10
Info in <TCanvas::Print>: pdf file ./plot/ROCs/ROC_10000_train_Classical.pdf has been created
Info in <TCanvas::Print>: pdf file ./plot/ROCs/ROC_10000_val_Classical.pdf has been created
Epoch: 10 | Epoch Time: 0m 50s
        Train Loss: 0.485 | Train Acc: 76.39%
         Val. Loss: 0.467 |  Val. Acc: 77.66%
```



Qiyu Sha qsha@cern.ch

Quantum transformer

- Zc(3900) decay chain
  $e^+e^- \to Zc(3900)^{\pm}\pi^{\mp}$
  $Zc(3900)^{\pm} \to J/\psi\pi^{\pm}$
  $J/\psi \to e^+e^-(\mu^+\mu^-)$
- Signal MC sample (BOSS 7.0.3)

decay tree	**decay model**
$e^+e^- \to Zc(3900)^{\pm}\pi^{\mp}$	PHSP
$Zc(3900)^{\pm} \to J/\psi\pi^{\pm}$	PHSP
$J/\psi \to e^+e^-(\mu^+\mu^-)$	VLL

- Data Samlpe (BOSS 7.0.3)
  4.260 GeV Data at BESIII of 2013

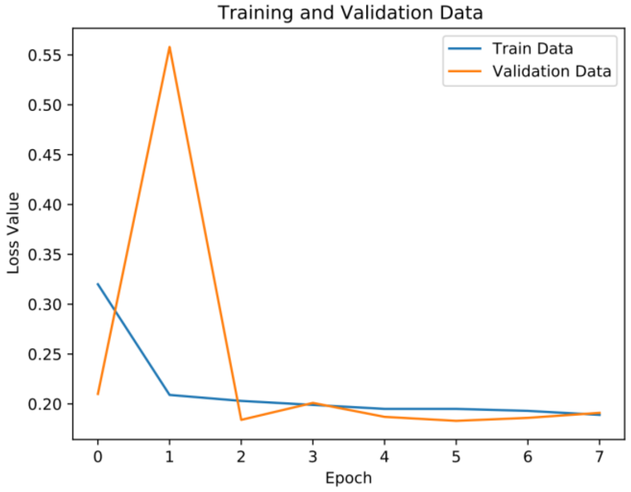
sample	**luminaance**	**center-mass energy**	**Run number**
4260	828.4±0.1±5.5	4257.97±0.04±0.66	29677-30367 31561-31981

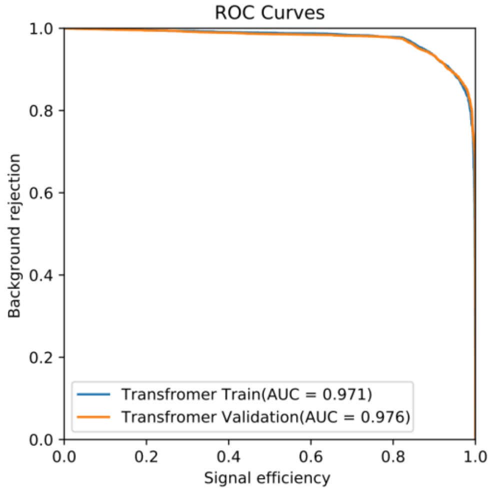
Preliminary Selection

- Four good tracks, zero net charge
- π: $p < 1$ Gev/c
- lepton :$p > 1$ Gev/c
  - e : $p > 1$ Gev/c and $E_{EMC} > 0.8 GeV$
  - μ: $p > 1$ Gev/c and $E_{EMC} < 0.6 GeV$
  - The number of pions and leptons should be two in each event with zero net charge.
  - $E_{EMC} > 1.1$ GeV identified as e
  - $E_{EMC} < 0.35$ GeV identified as μ
- remove gamma-conversion background
  - $\cos(\pi^+\pi^-) < 0.98$
  - $\cos(\pi^{\pm}e^{\mp}) < 0.98$

Quantum transformer

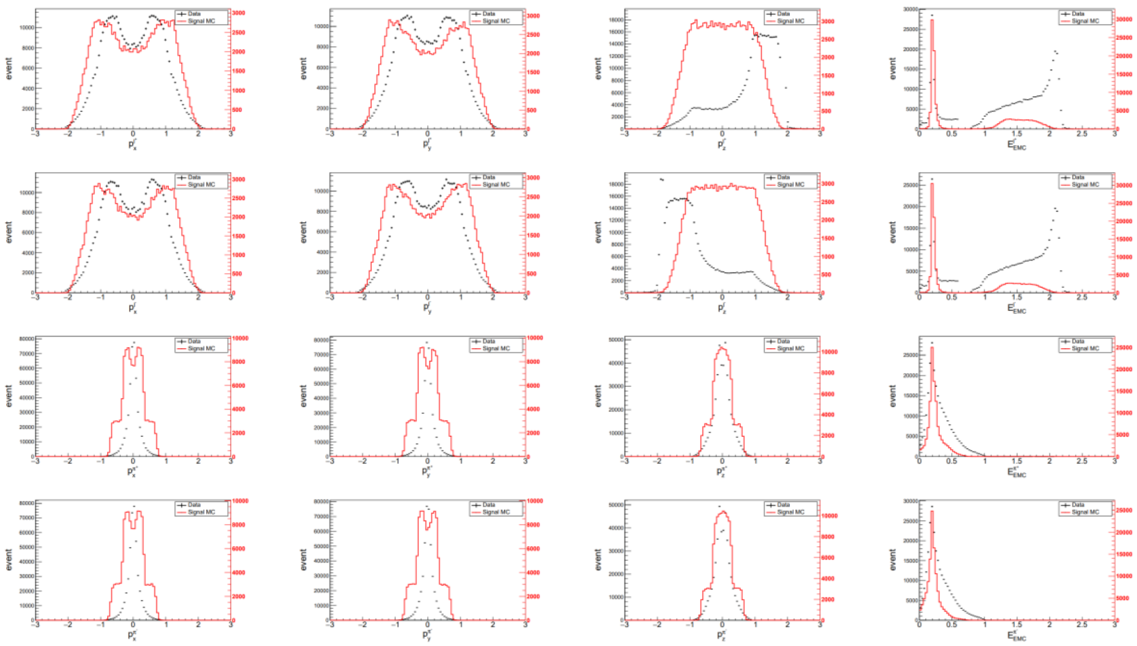
- Model : Quantum Transformer
- Parameter Set:17 variables
  - $\vec{p}, E, \chi^2$(4C kinematic Fit)
- Signal : Signal MC
- Background is form data : $M(l^+l^-) \in (0, 3.06) \bigcup (3.14, 5)$ GeV$/c^2$
- Date Set :
  - 20k events(10k for train, 10k for validation)
  - 8 epochs

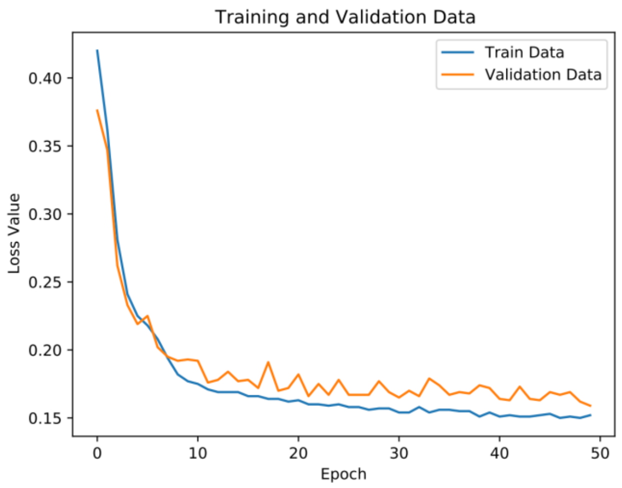


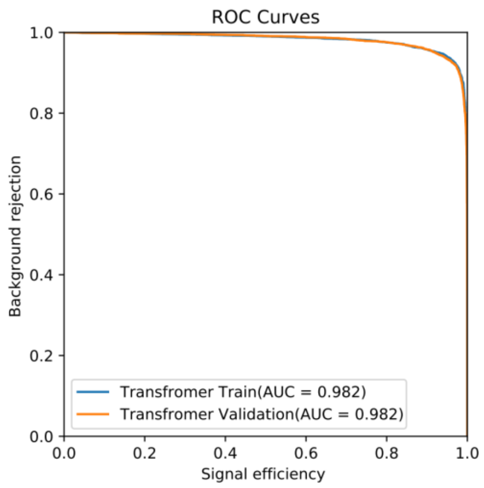


Transformer

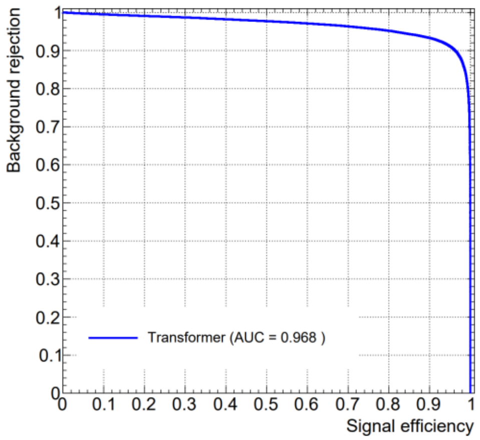
- Model : Transformer
- Parameter Set:26 variables
  - $\overrightarrow{p}, E, p, \theta_{l^\pm\pi^\mp}, \theta_{l^+l^-}, \theta_{\pi^+\pi^-}, N_\gamma, E_\gamma^{max}$

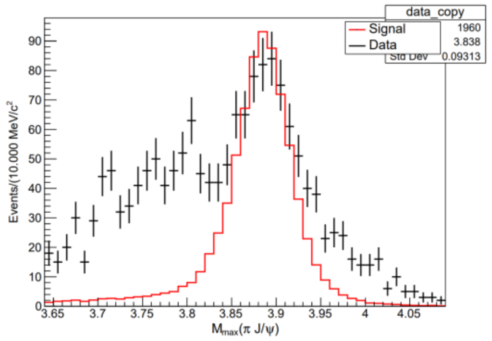






Apply Model





Data : 1960

Signal : 60789