



Quantum C2C

Deep learning for compression of classical data in quantum computing

TEAM PRESENTATION

**Chih-Han (Robin)
Huang**

Data Scientist

Deep Learning, Quantum
Computing

Sumitra Pundlik

Professor

Quantum Computing, Data
Science

Khushwant Kumar

Engineer

Deep learning, Quantum
Computing

Vardaan Sahgal

Physics Master Student

Physics,, Quantum Computing

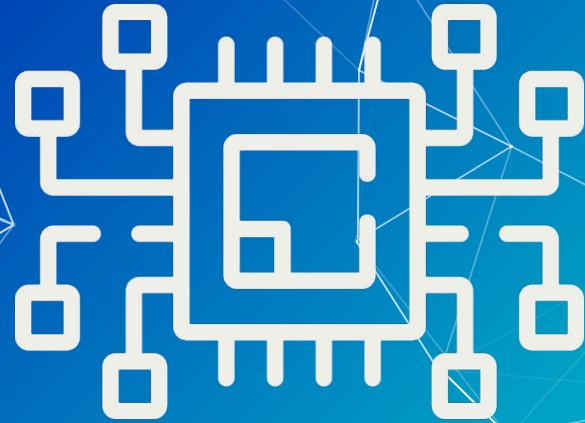
Tathagata Majumdar

Computer Science Bachelor Student

Computer Science, Quantum
Computing

Data compression of classical data is important

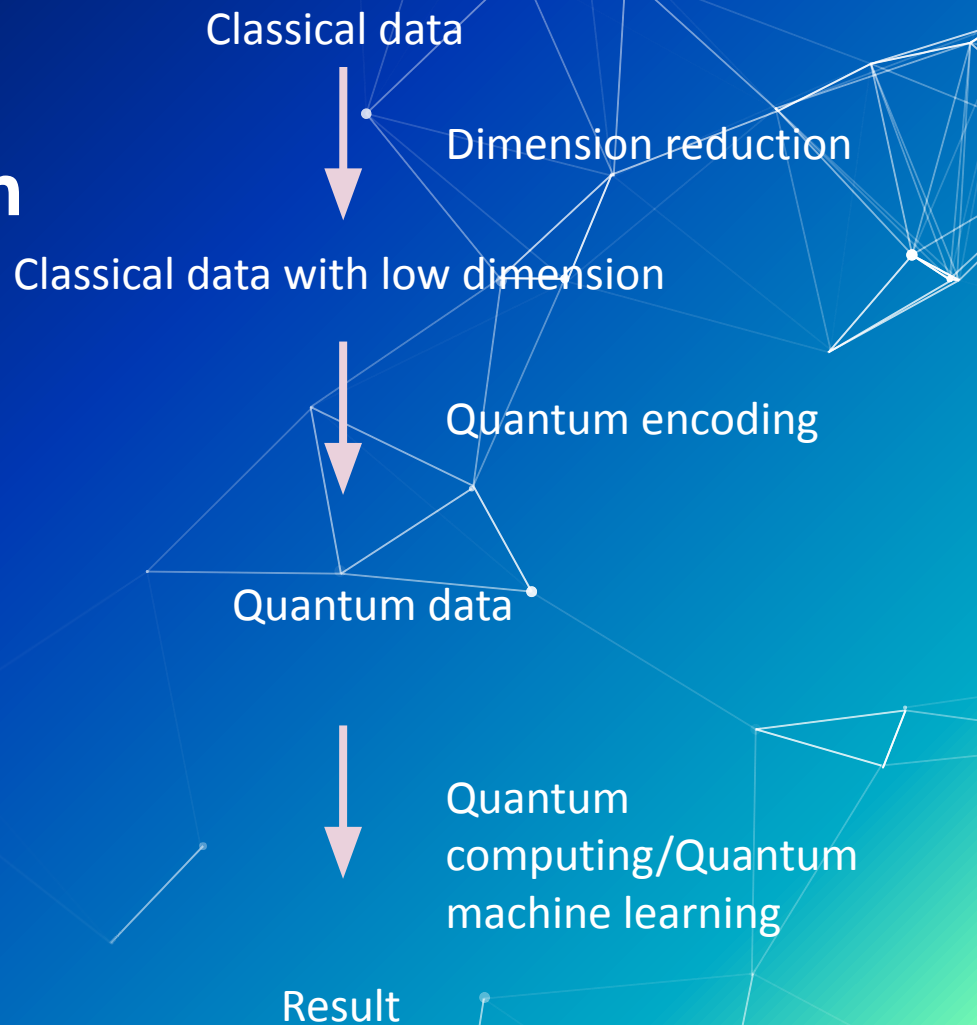
In a noisy intermediate-scale quantum era



Classical data compression

In previous literatures about quantum computing (QC) or quantum machine learning (QML)

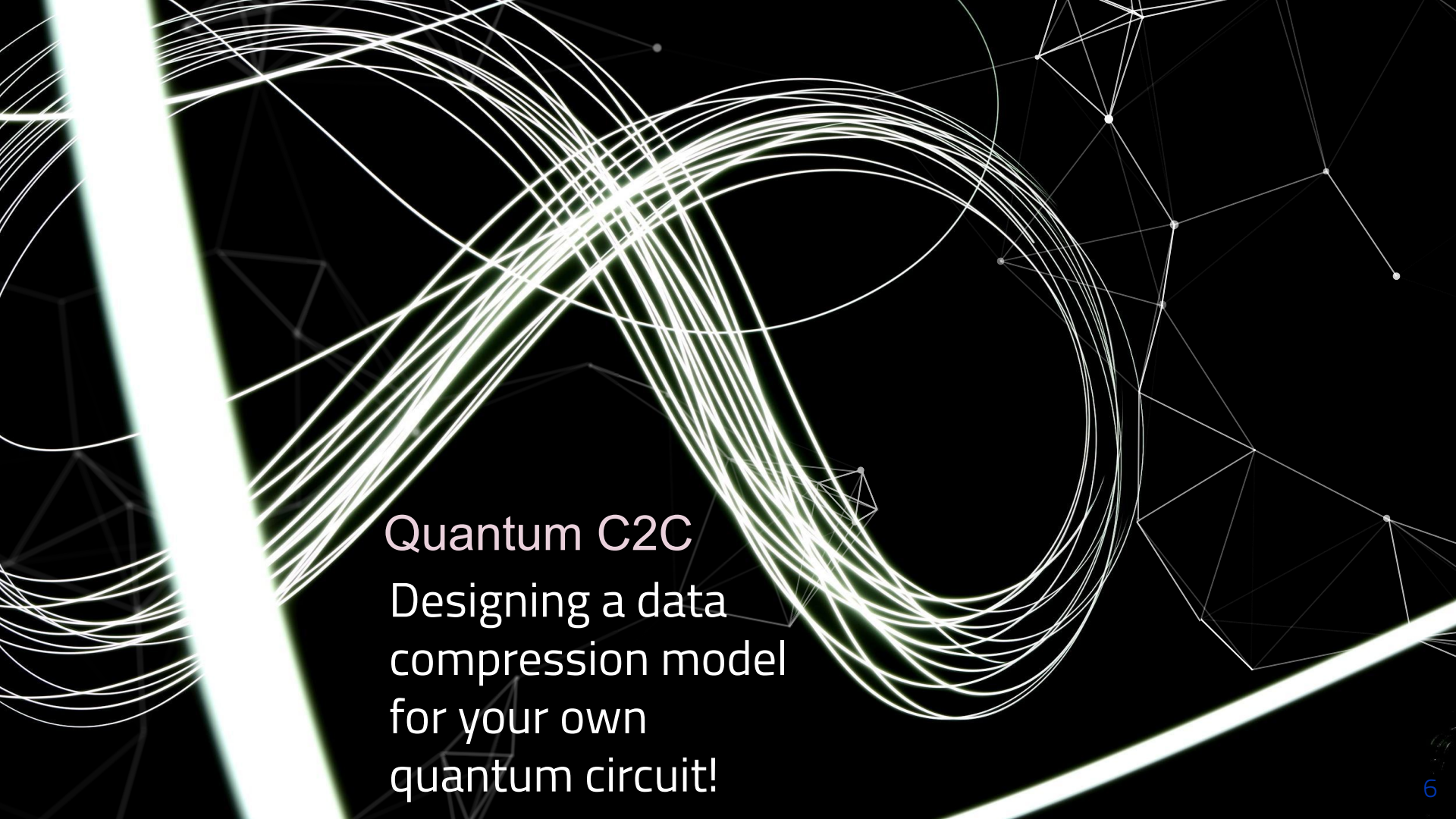
- Simple or non-parametric
 - Downsampling image resolutions
 - Principal components analysis



**Suitable for any
quantum circuit?**

Information loss?





Quantum C2C
Designing a data
compression model
for your own
quantum circuit!

Main idea

- We hypothesized that deep learning (DL) would learn the optimized parameters to compress classical data for QC/QML.
- Information loss would be minimized during the data compression with deep learning.
- We argued that each quantum circuit would be suitable for different data compression models (both hyperparameters and parameters).
- One could train and design different DL data compression model structures for several quantum circuits.

Quantum C2C methodology

We first proposed to use Multitask Learning on QC/QML to simultaneously minimize the loss of autoencoder and loss of performance of QC/QML.

Steps

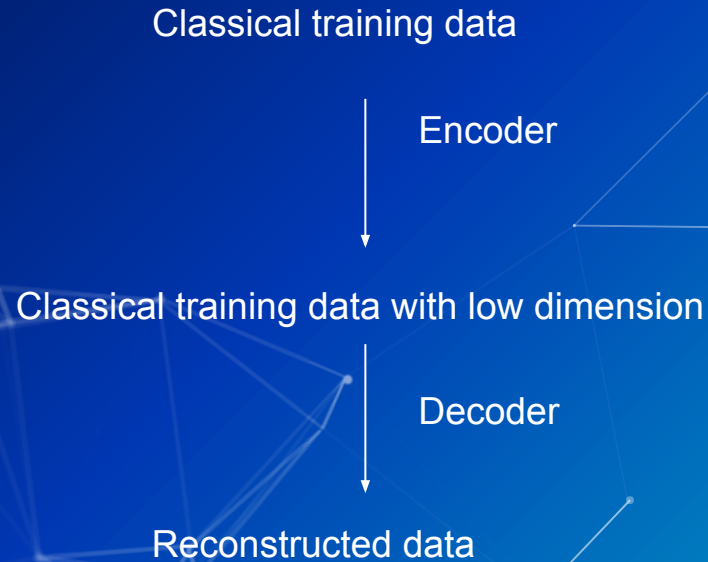
1

Autoencoder
pre-training

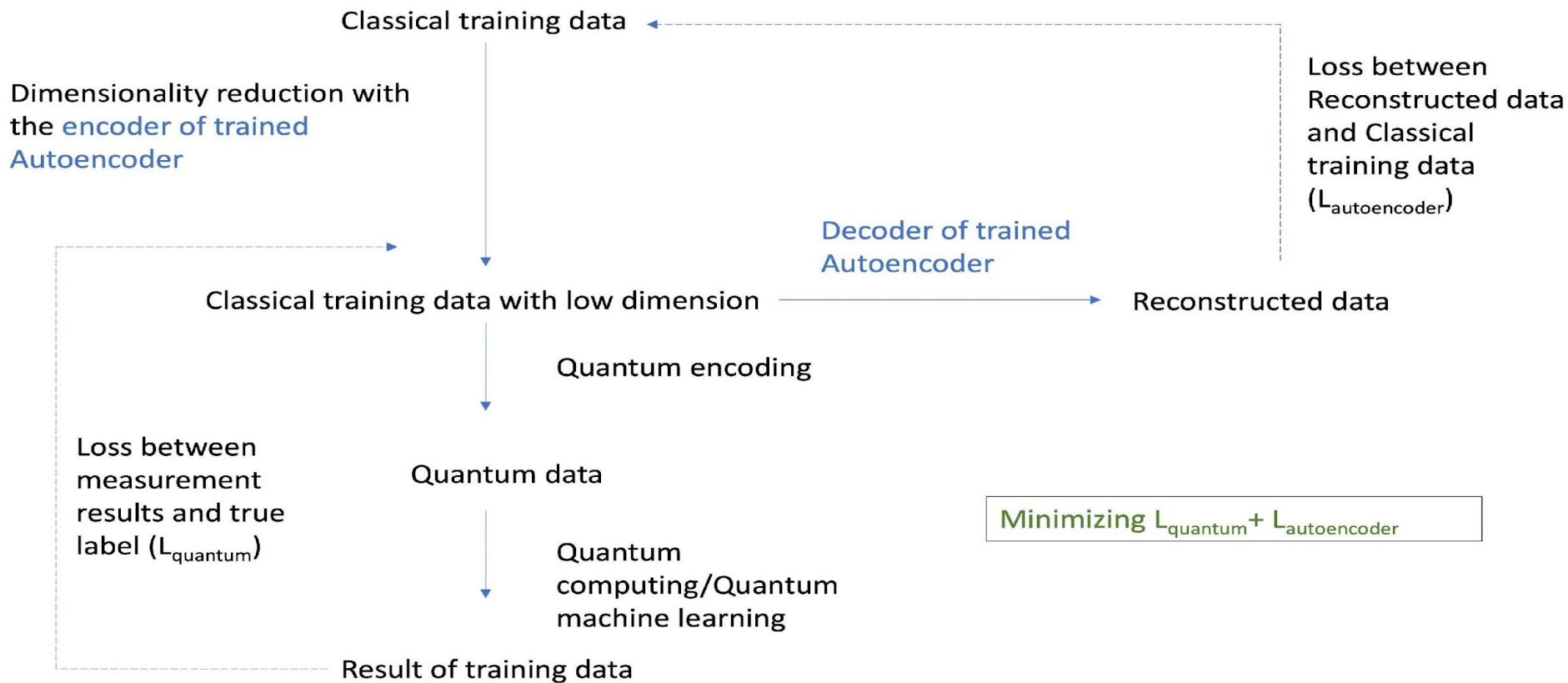
2

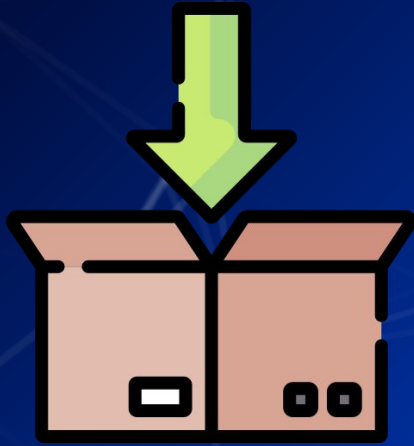
Main training

Autoencoder pre-training



Main training





**A tool of Quantum C2C for
QML is also developed**

Install

```
cd lib  
python setup.py install
```


Usage

```
from quantum_c2c.quantum_c2c import quantum_c2c, AutoEncoder, Hybrid
```

```
ae=AutoEncoder(input_shape=X_train.data.shape,encoded_len=encoded_len)  
qc=Hybrid(qiskit.Aer.get_backend('aer_simulator'),  
          100, np.pi / 2,encoded_len=encoded_len)
```

```
trained_encoder_model,y_predicted=quantum_c2c(X_train_autoencoder,  
                                                X_train,  
                                                X_test,  
                                                autoencoder_model=ae,  
                                                quantum_circuit=qc,  
                                                saving_folder='model',  
                                                epochs=epochs,  
                                                encoded_len=encoded_len)
```

Input

- **a. X_train_autoencoder:**
 - Classical Data for the autoencoder pre-training.
- **b. X_train:**
 - Classical Data for main model training.
- **c. X_test:**
 - Classical Data for main model testing.
- **d. Autoencoder_model:**
 - PyTorch model of autoencoder.
- **e. Quantum_circuit:**
 - Qiskit's "Quantum-Classical Class" with PyTorch
- **f. Saving_folder:**
 - Folder name for saving the models "pretrained_autoencoder.pth" and "trained_encoder_model.pth".
- **g. Epochs:**
 - Epochs for the autoencoder pre-training and main model training.
- **h. Encoded_len:**
 - length of encoded data

Output

- **a.model.encoder:**
 - Return of the function. The last trained encoder model
- **b. y_predicted:**
 - Return of the function. Predicted target values
- **c. pretrained_autoencoder.pth:**
 - The file saving in saving_folder. Pretrained autoencoder.
- **d. trained_encoder_model.pth:**
 - The file saving in saving_folder. Trained encoder model.



Thank you for listening