Application of Quantum Machine Learning in High Energy Physics Experiments

Xingtao Huang
Institute of Frontier and Interdisciplinary Science
Shandong University

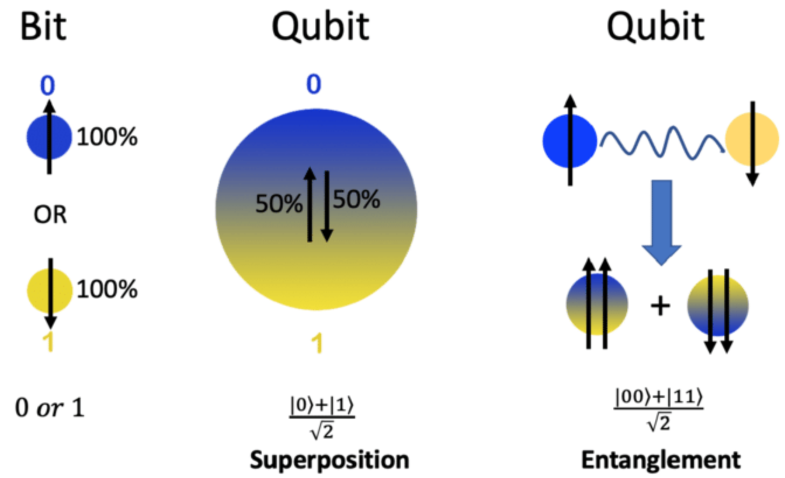
量子计算与高能核物理前沿交叉讲习班
华南师范大学
2022年11月13-28日

Outline

- Introduction to QC and QML
- CERN Quantum Technology Initiative
  - Strategy and Roadmap
  - Applications of QML in HEP
- Activities of SDU+IHEP QML Group
  - Quantum Support Vector Machine
  - Variational Quantum Classifier
- Summary
- QML Tutorial

Quantum Computing (QC)

- Qubits are used by quantum computers to measure and extract information
  - Superposition
  - Entanglement



https://www.researchgate.net/publication/344971320

- **Advantages of QC**
  - speed up

    Google*: 53 qubits /200 s ~ 2^{53} (10^{16}) dimensions/ 10000 year
  - solve complex problems
  - run complex simulations

*(*Nature* **volume 574**, pages 505–510 (2019))

Quantum Machine Learning (QML)

- HEP accumulates larger volumes of data, and requires higher and higher precision.
  - Migrated towards ML in past 20-30 years
  - Challenges of ML : heavy CPU time, global optimization ? ...
- Quantum computer provides a new set of tools for ML
  - Serve as a valuable alternative for classical ML models
  - Provide more efficient computing devices
- Potential quantum advantage for ML problems
  - Potential speed-up for training [1]
  - Data is processed in a high dimensional Hilbert spaces that is intractable on classical computers [2]

CERN Quantum Technology Initiative (CERN QTI)


QUANTUM TECHNOLOGY INITIATIVE

CERN
Quantum Technology
Initiative

Strategy and Roadmap

v. 1.0-Rev2

7 October 2021

CERN-OPEN-2021-012
07/10/2021

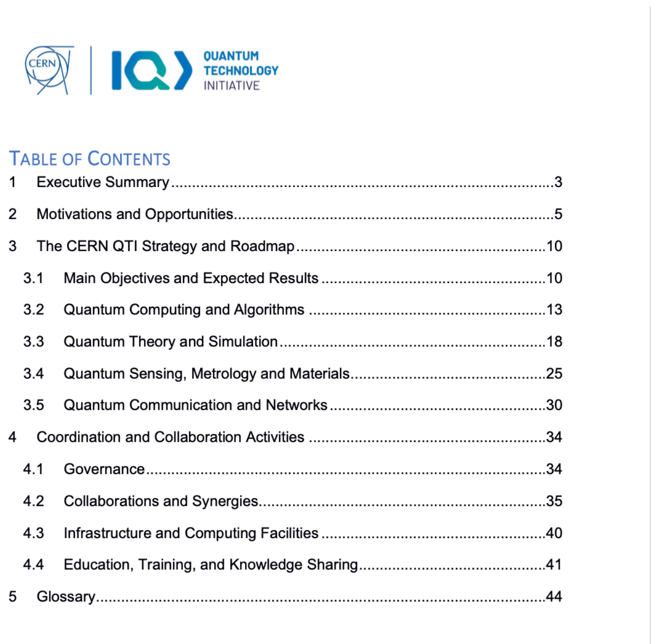
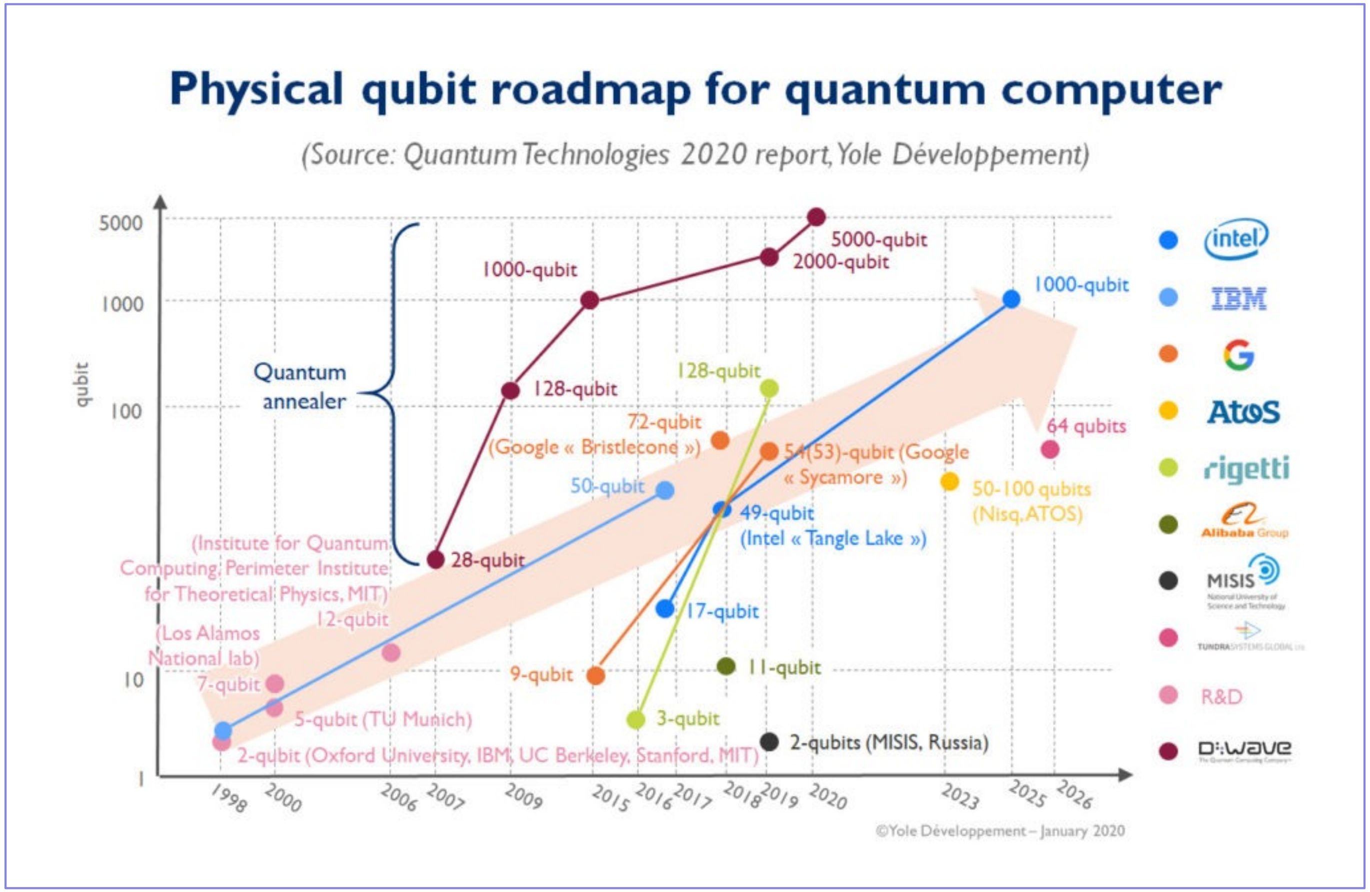

QUANTUM TECHNOLOGY INITIATIVE

TABLE OF CONTENTS

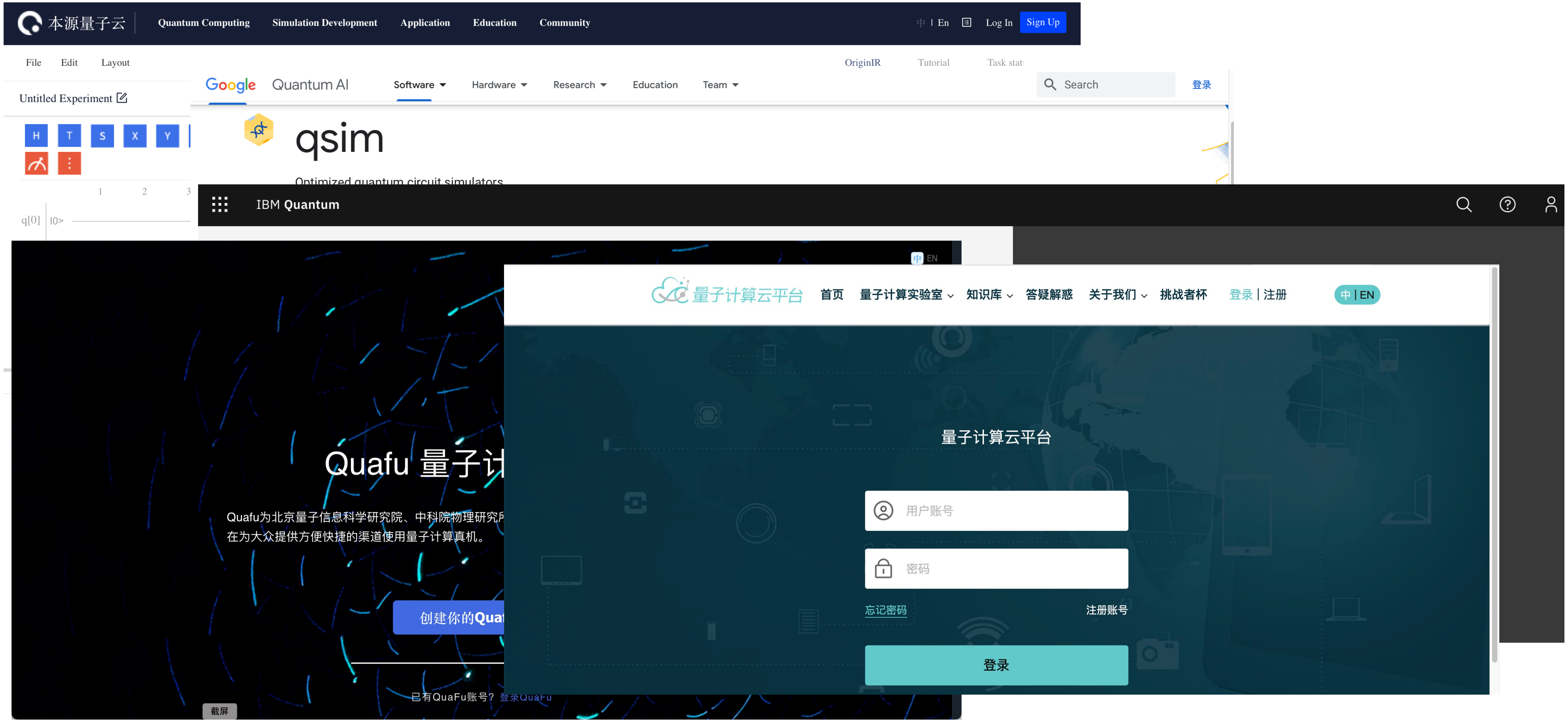
Activities in Quantum Computing and Algorithms

- Quantum Algorithms for HEP workloads and QML
- Algorithm Optimization and Benchmarking
- Design of a distributed infrastructure for Quantum Computing



Activities in Quantum Computing and Algorithms

- Quantum Algorithms for HEP workloads and QML
- Algorithm Optimization and Benchmarking
- Design of a distributed infrastructure for Quantum Computing

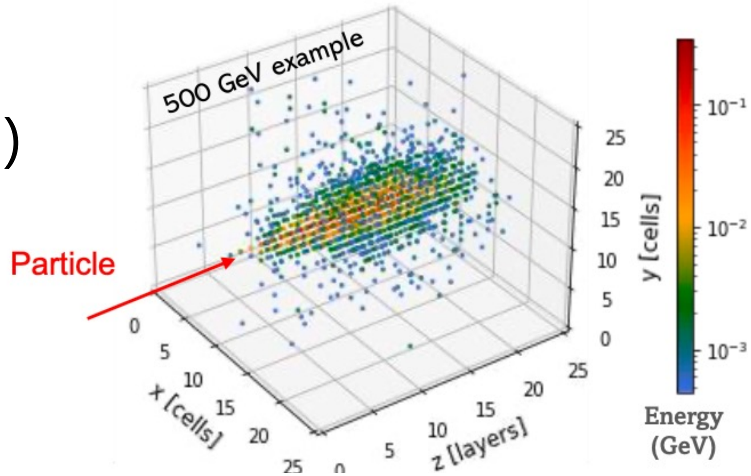


Quantum Algorithms for HEP workloads and QML

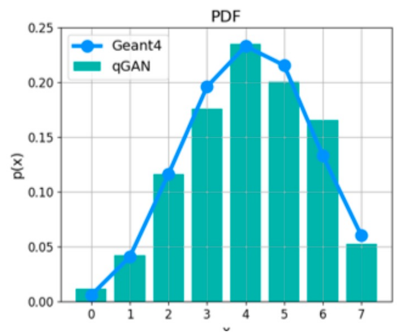
- **Quantum Generative Adversarial Networks for detector simulation**
  - *npj Quantum Information* **volume 5**, Article number: 103 (2019)
- **Quantum Graph Neural Network algorithm for tracking**
  - Quantum Machine Intelligence (2021) 3: 29, https://doi.org/10.1007/s42484-021-00055-9
- **Event Classification with QML in High-Energy Physics**
  - Computing and Software for Big Science (2021) 5:2, https://doi.org/10.1007/s41781-020-00047-7
- **Quantum SVM for Higgs classification**
  - Phys. Rev. Res. 3 (2021) 3, 033221
- **Quantum Machine Learning for b-jet charge identification**
  - J. High Energ. Phys. 2022, 14 (2022)
- **Quantum Convolutional Neural Networks for Event Classification**
- **Quantum optimization for grid computing , …….**

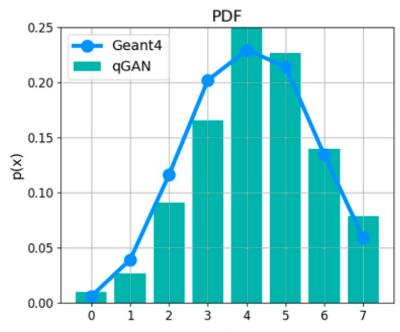
QGAN for Calorimeter Simulation

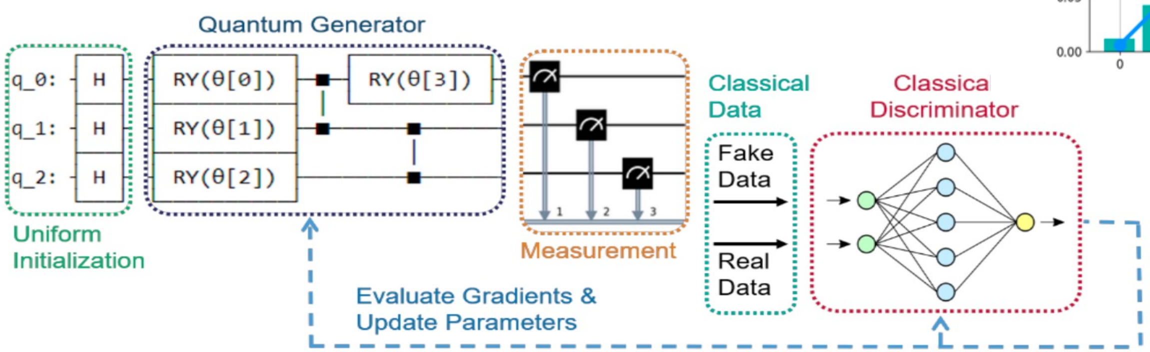
- Calorimeter simulation with Geant4 is the most time-consuming part
- DLGAN gains speed up (up to 160000 x)
- Hybrid quantum-classical GAN (10 x)



- Modified a Qiskit qGAN model
  - Simplified model :1D 8-pixel images
  - Amplitude encoding: 3 qubits (2^3 = 8 states)

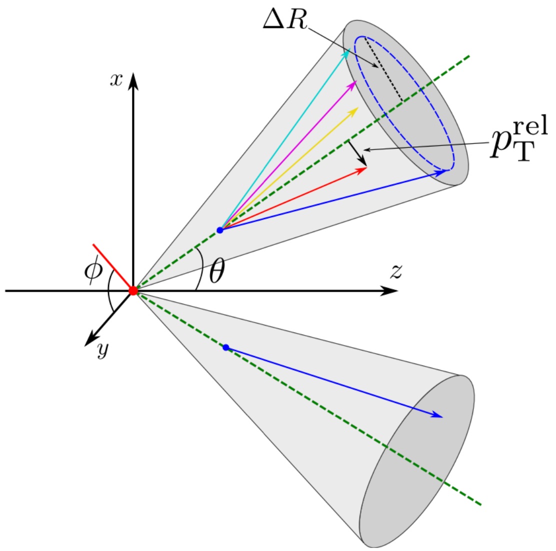




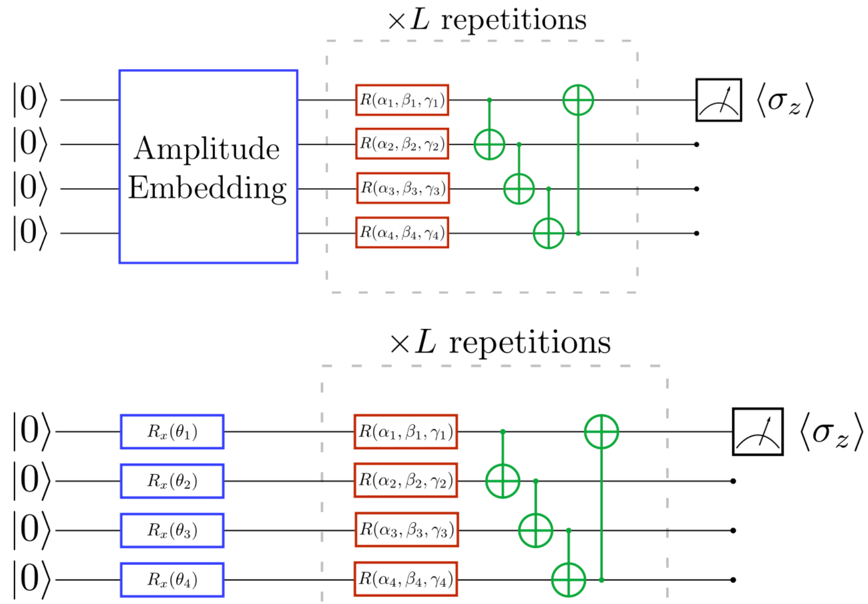


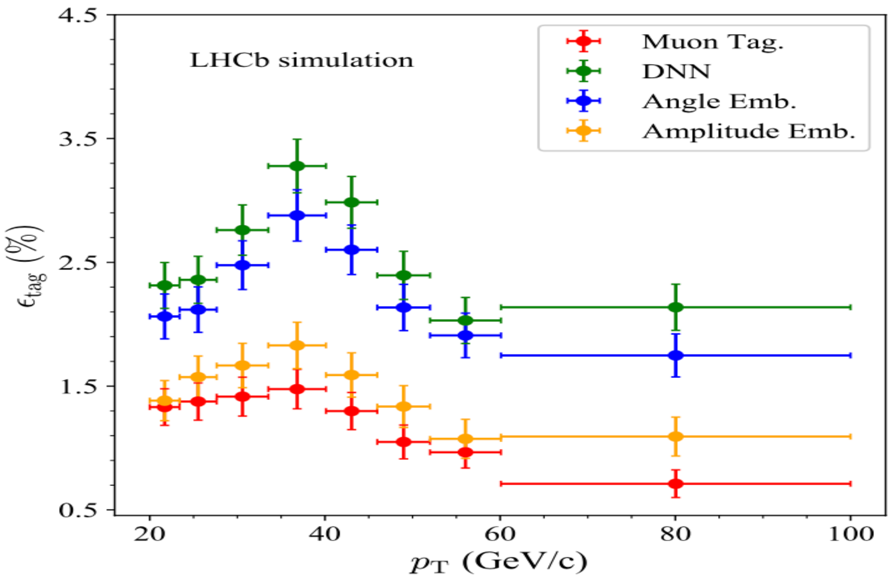
https://ceur-ws.org/Vol-3041/363-368-paper-67.pdf

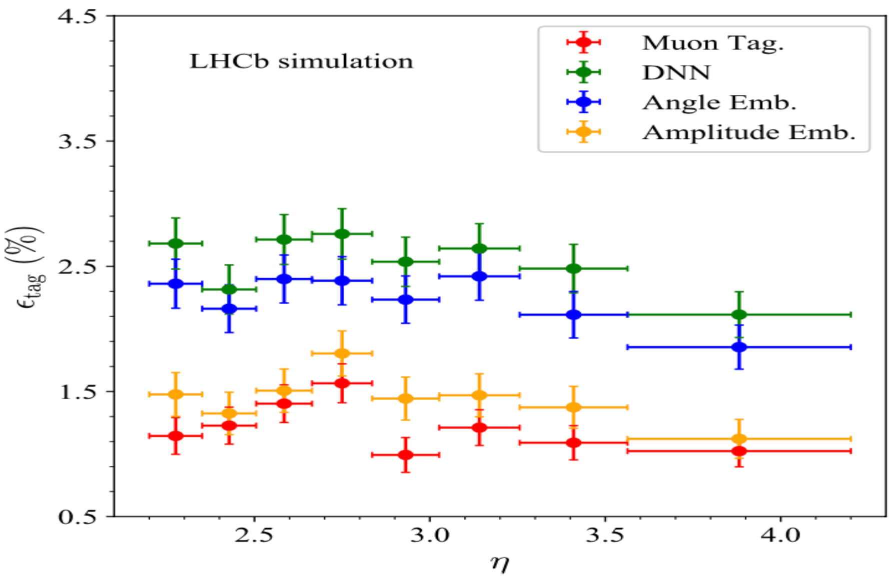
Variational Quantum Classifier for b-jet charge identification



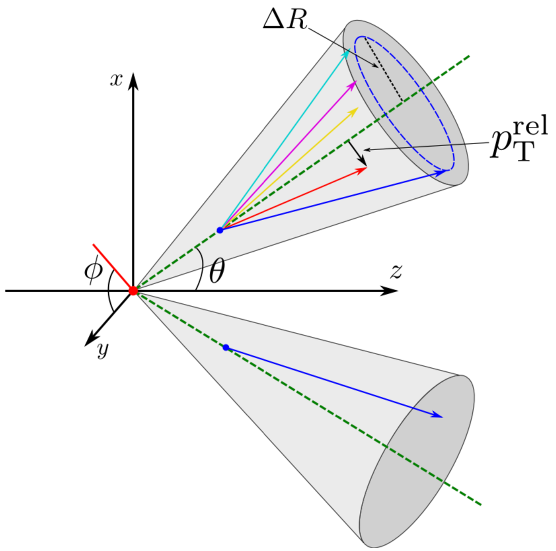
J. High Energ. Phys. 2022, 14 (2022)



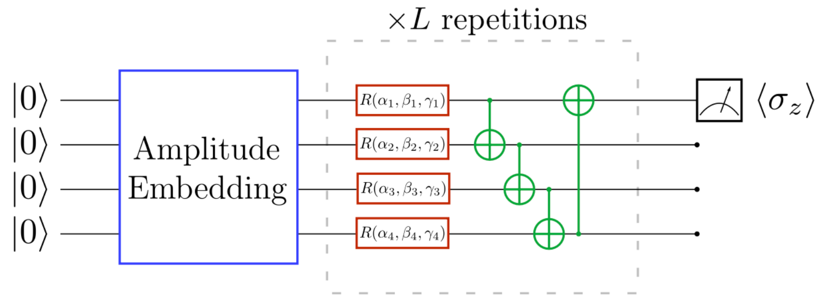


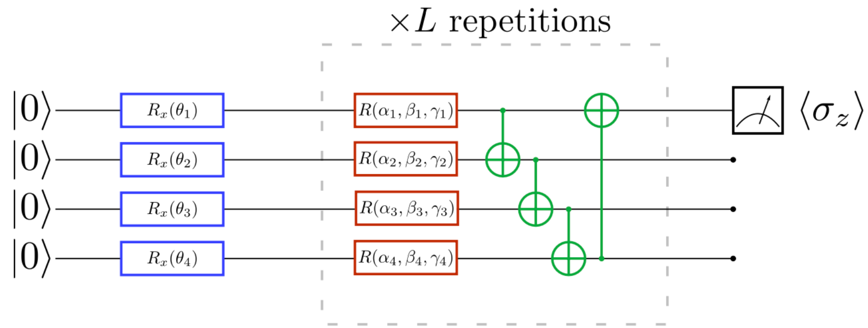


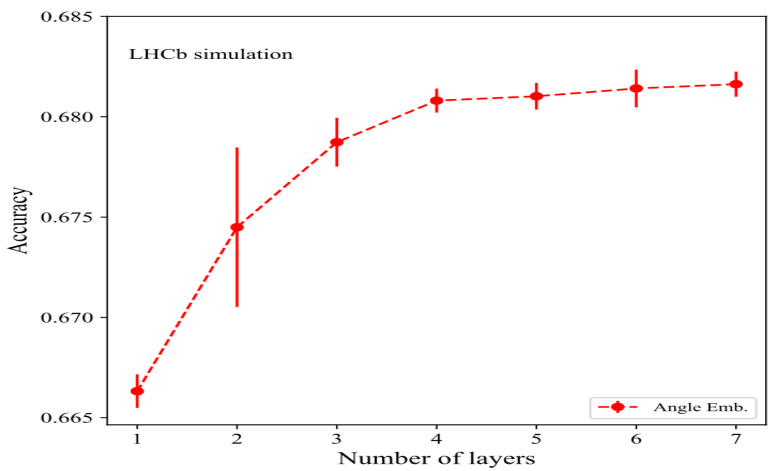
Variational Quantum Classifier for b-jet charge identification

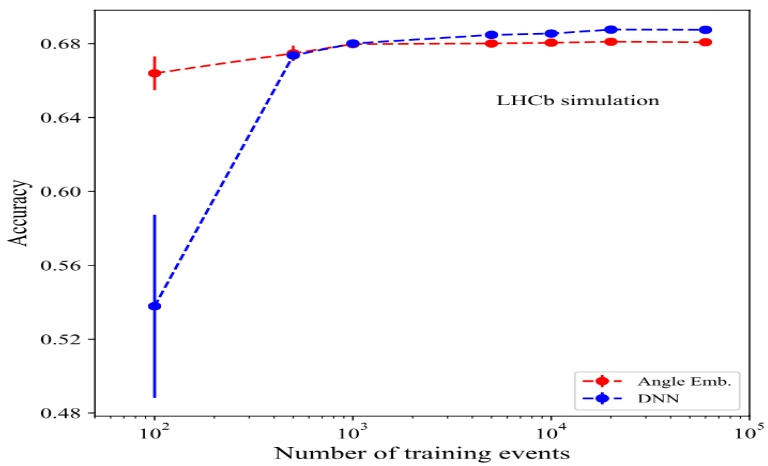


J. High Energ. Phys. 2022, 14 (2022)





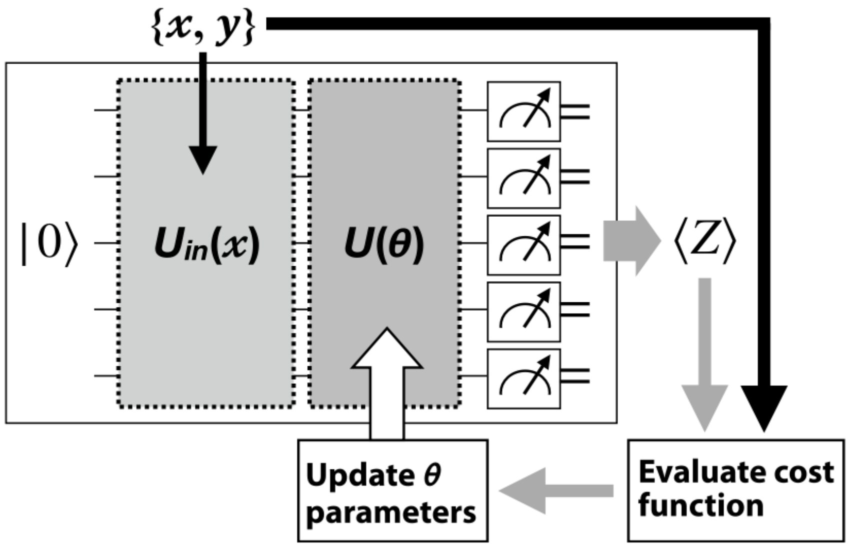


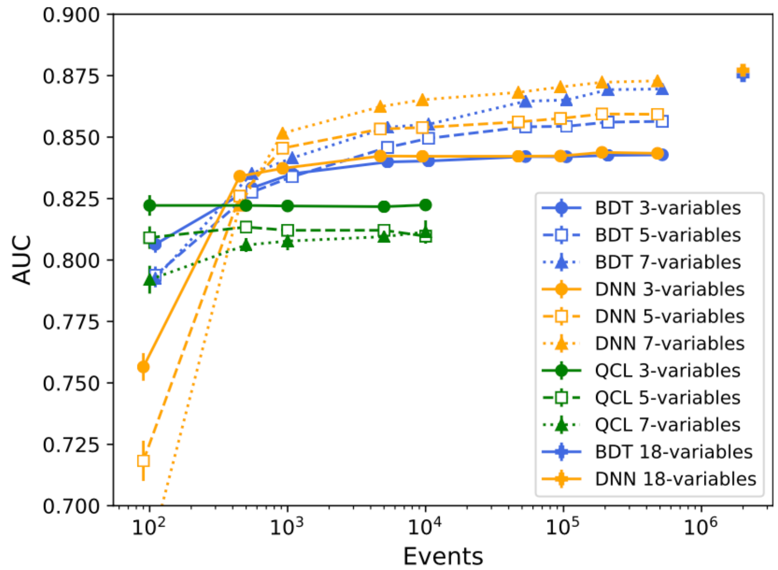


Event Classification with Quantum Circuit Learning (QCL)

- Most frequently used ML technique in HEP data analysis is the discrimination of events of interest

  E.g. signal events from background events （SUSY /WW ）

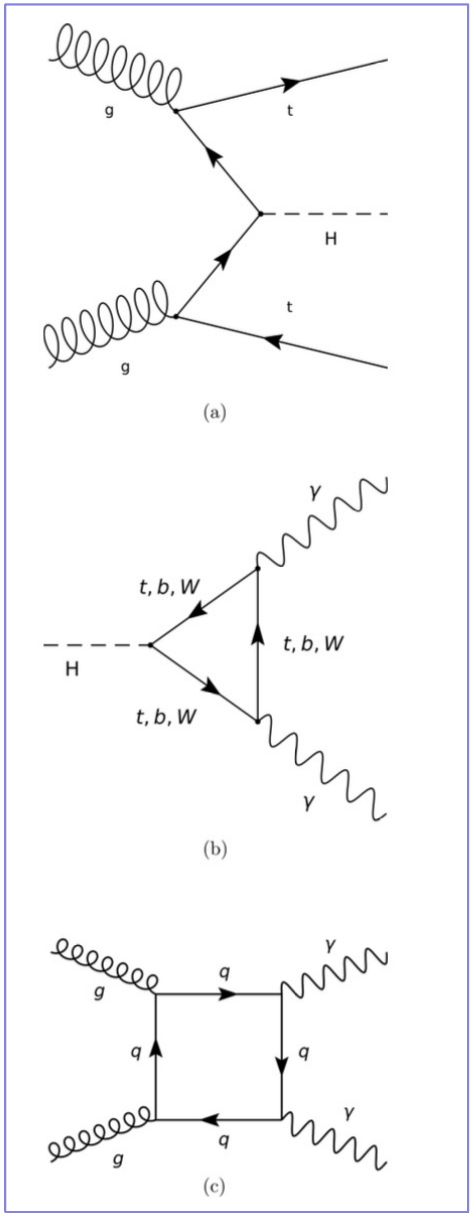


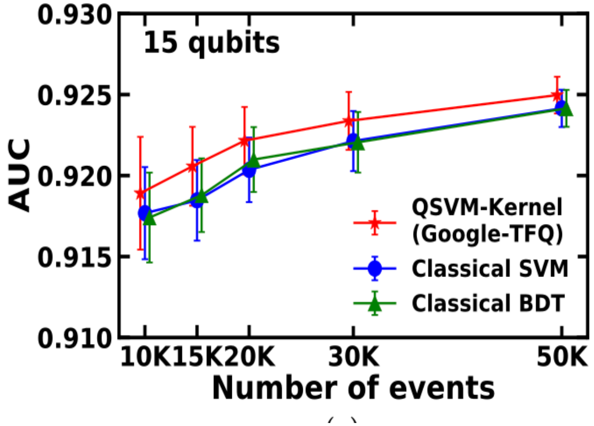


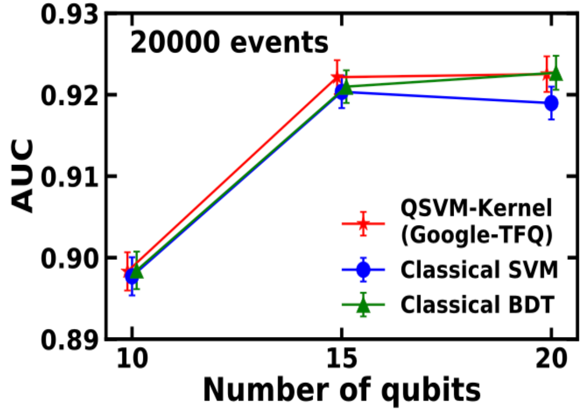
- The QCL performance is comparable to BDT and DNN
  - 3 variables is enough to discriminate signal from background
  - the performance of the BDT and DNN improve rapidly with number of events

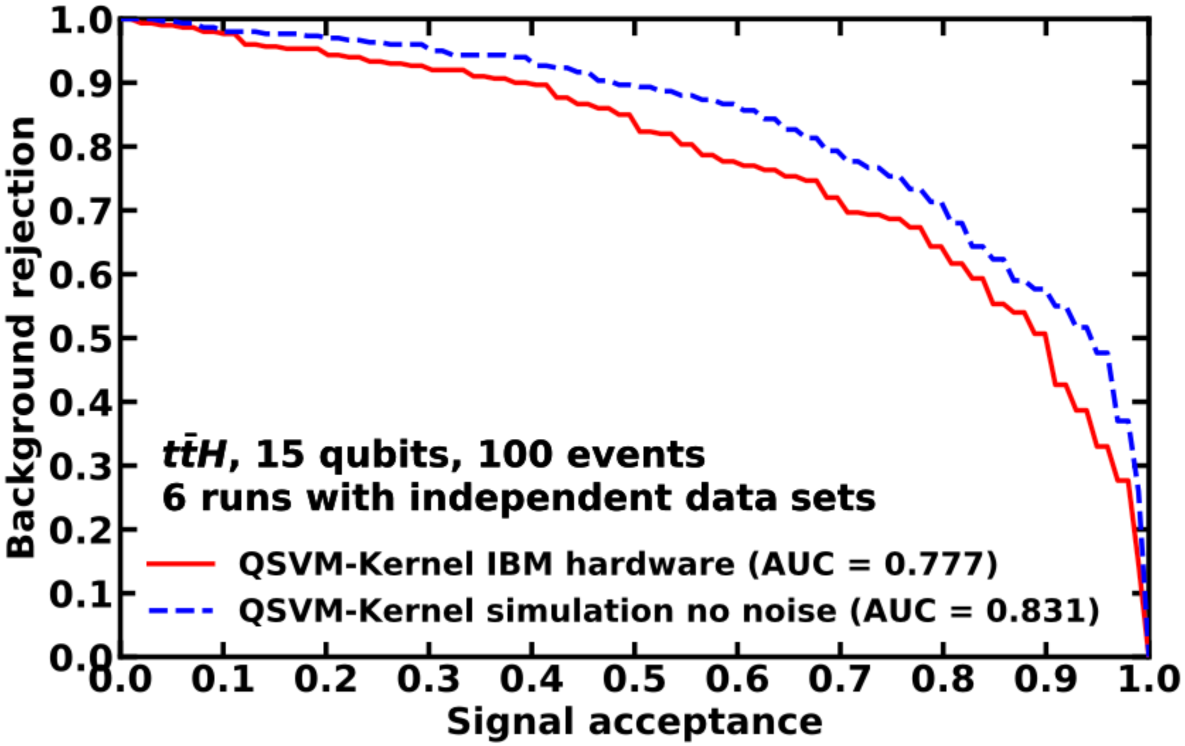
Computing and Software for Big Science (2021) 5:2
https://doi.org/10.1007/s41781-020-00047-7

Employing QSVM-Kernel for ttH (H-> $\gamma\gamma$) analysis at ATLAS







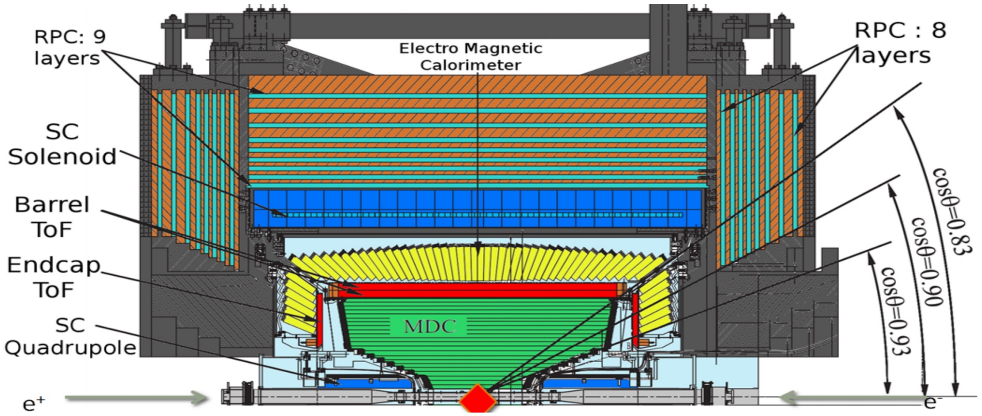


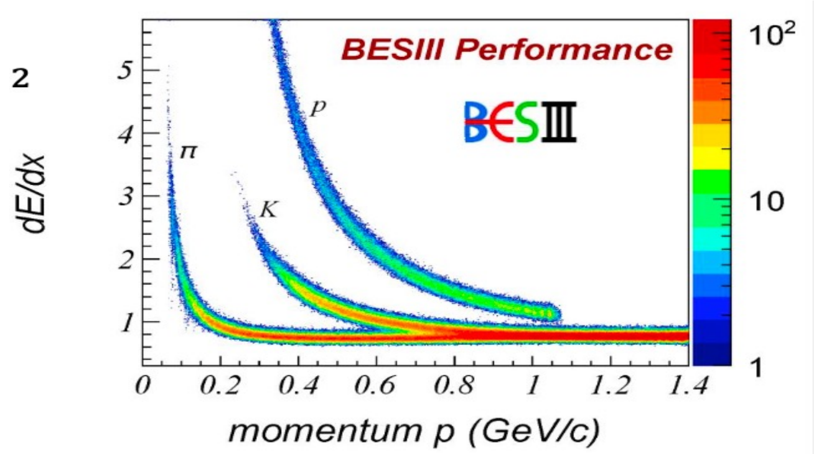
Our Program with Quantum Machine Learning

- Started in Nov. 2020
- Manpower
  - SDU：Teng Li, Zhipeng Yao, Xingtao Huang
  - IHEP：Jiaheng Zou, Tao Lin, Weidong Li
- QML application in PID, tracking and Analysis
  - Variational Quantum Classifier Method
  - Quantum Support Vector Machine Kernel Method
  - Quantum Neural Network Method
- Study quantum computing as a proof of concept
  - Test Under Noisy Intermediate-Scale Quantum (NISQ) device
  - Explore and demonstrate of the potential of quantum computer in HEP experiments [3-5]
  - Pave the way for future applications (e.g. analysis, tracking, …)

Particle Identification at BESIII

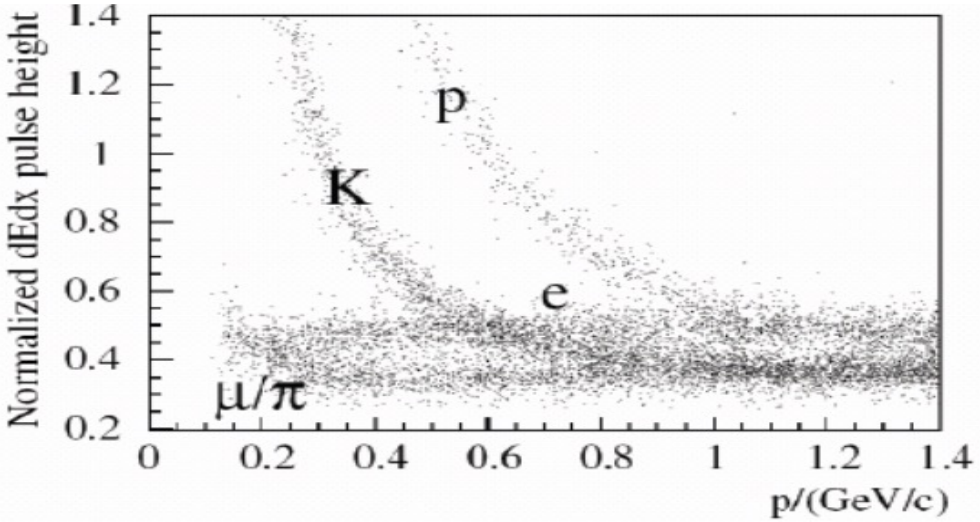
- **PID performance is critical for various physics studies at BESIII**

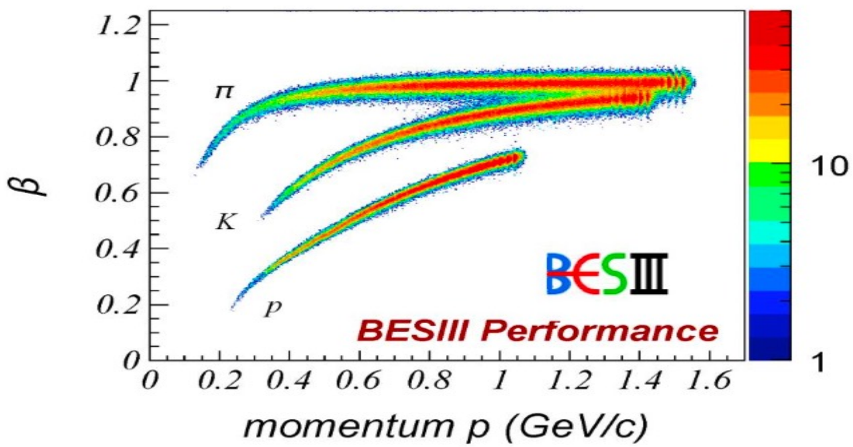




$$\chi_{dE/dx} = \frac{(dE/dx)_{mea} - (dE/dx)_{exp\_i}}{\sigma(dE/dx)}$$

$$\chi_{comb} = \chi_{dE/dx} + \chi_{TOF}$$

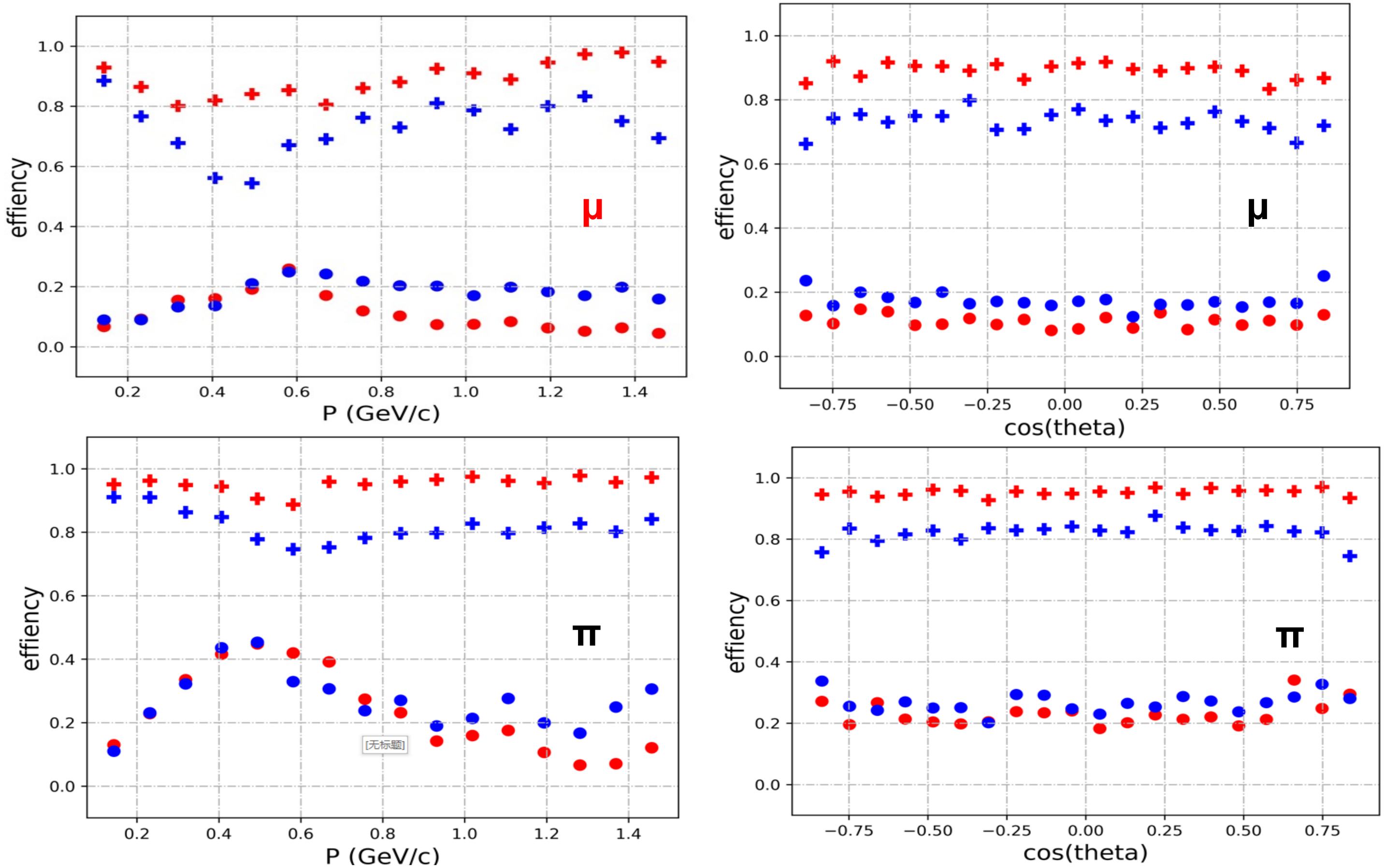




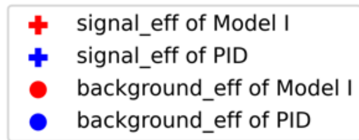
$$\chi_{TOF} = \frac{(TOF)_{mea} - (TOF)_{pred\_i}}{\sigma(TOF)}$$

Particle Identification with ML

- **Machine learning has armed PID with a powerful toolbox**
  - Frequently used models include SVM, DNN, CNN, BDT and MLP etc.
  - Good at combining information of multiple sub-detectors, especially for hard PID tasks (such as, μ/π separation in this study)



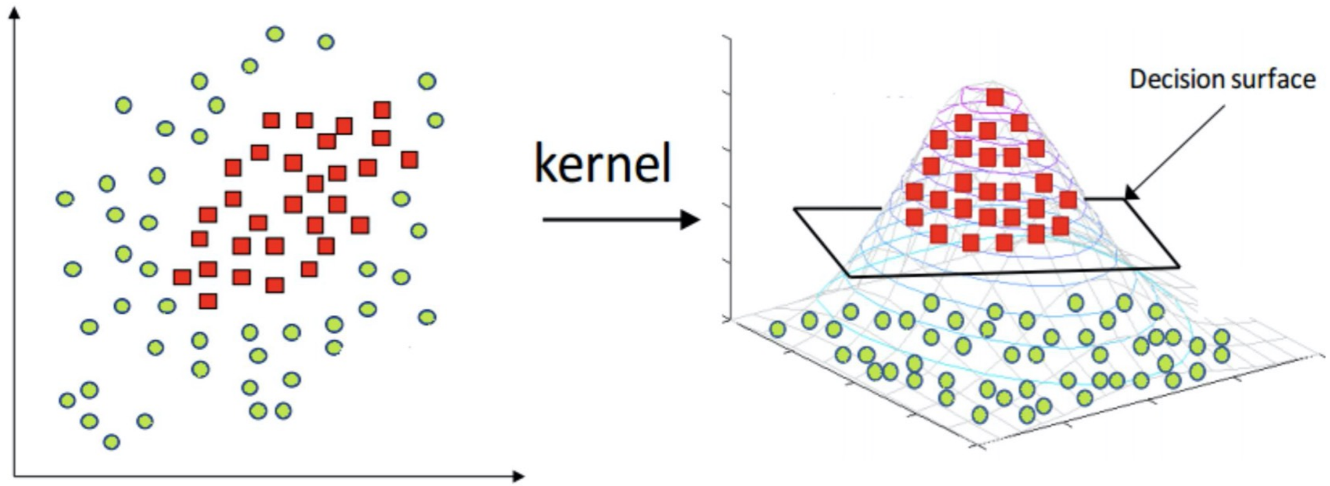
- XGBoost
- Selected 47 out of 108 features



Method I
Quantum Support Vector Machine

Classical Support Vector Machine

- Support Vector Machine (large margin classifier)



- The heavy part of training SVM is the computation of the kernel matrix

$$\text{maximize} \quad L(\vec{\alpha}) = \sum_{i=1}^{N} y_i \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \boxed{K(\vec{x}_i, \vec{x}_j)}$$

$$\text{subject to} \quad \sum_{i=1}^{N} \alpha_i y_i = 0 \text{ and } \alpha_i \geq 0, \forall i = 1, 2, ..., N$$

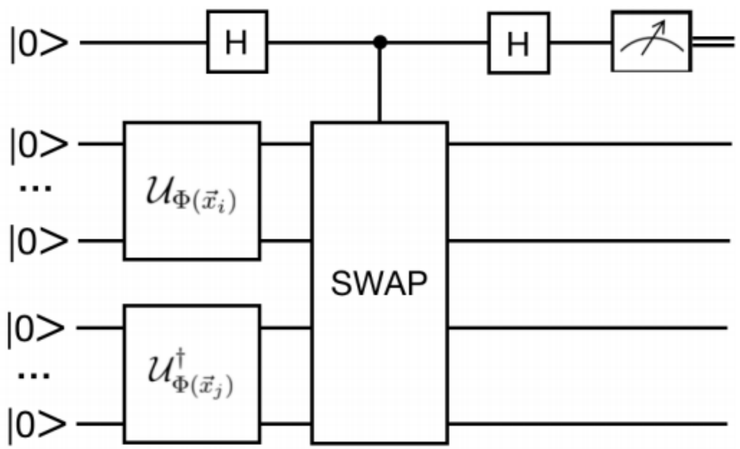
- Quantum device provides an extension of the kernel methods

Quantum Support Vector Machine

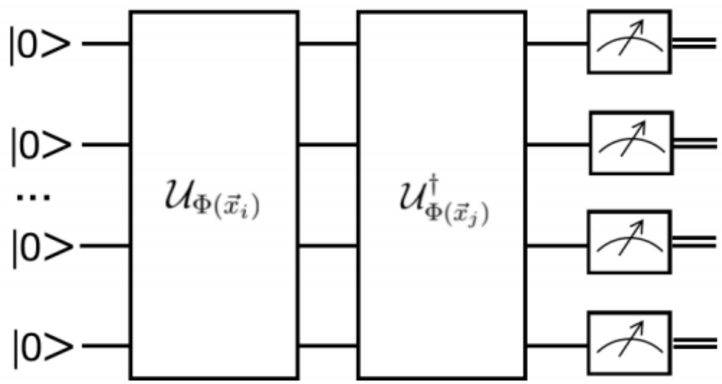
- The inner product of two quantum states representing two data points can be seen as the kernel [6]

$$K(\vec{x}_i, \vec{x}_j) = |\langle \Phi(\vec{x}_i) | \Phi(\vec{x}_j) \rangle|^2 = \left| \langle 0^{\otimes n} | \mathcal{U}^\dagger_{\Phi(\vec{x}_j)} \mathcal{U}_{\Phi(\vec{x}_i)} | 0^{\otimes n} \rangle \right|^2$$

- The quantum circuits can estimate the kernel values:



(a)



(b)

Quantum Feature Map

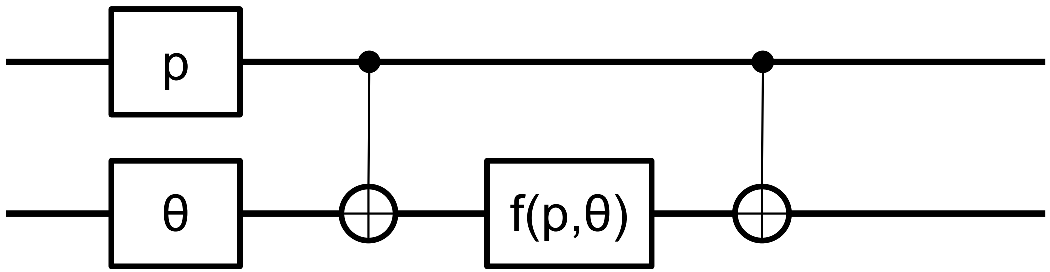
- The core part of QSVM is the encoding circuit (feature map) [7]

$$|\Phi(\vec{x}_i)\rangle = \mathcal{U}_{\Phi(\vec{x}_i)} \mid 0^{\otimes n}\rangle = H^{\otimes n} U_{\Phi(\vec{x}_i)} H^{\otimes n} U_{\Phi(\vec{x}_i)} \mid 0^{\otimes n}\rangle$$

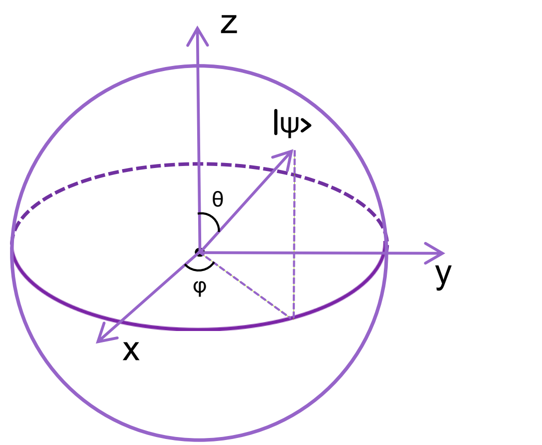
- The feature map encodes data points into the amplitude of quantum states based on Pauli rotation operators

$$U_{\Phi(\vec{x})} = exp(i \sum_{S \in [n]} \Phi_s(\vec{x}) \prod_{i \in S} P_i) \qquad \Phi_s(\vec{x}) = \begin{cases} x_i & \text{for the i-th qubit} \\ (\pi - x_i)(\pi - x_j) & \text{for the i-th and the j-th qubits} \end{cases}$$

Z,ZZ



$\boxed{x}$: rotate x angle around z axis



Training Sample and Baseline Models

- BESIII MC Sample:
  - Single $\mu^{\pm}$ and $\pi^{\pm}$ tracks from MC, 20000 training tracks and 10000 test tracks per dataset
  - Cross validation on 20 datasets
  - Nine selected features:
    - Reconstructed momentum and direction (p, θ)
    - PID likelihood from TOF and dE/dX ($\chi^2_{\mu\_tof}$, $\chi^2_{\pi\_tof}$, $\chi^2_{\mu\_\mathrm{dedx}}$, $\chi^2_{\pi\_\mathrm{dedx}}$)
    - Shower shape in EMC($\frac{E_{3\times3}}{E_{seed}}$, $\frac{E_{5\times5}}{E_{3\times3}}$)
    - Penetration depth in MUC (depth)
- Baseline models are carefully tuned as control group
  - Classical SVM: scikit-learn 0.24.1
  - BDT: py-xgboost 0.90
  - MLP: tensorflow 2.4.1

Scan of Various Encoding Circuits

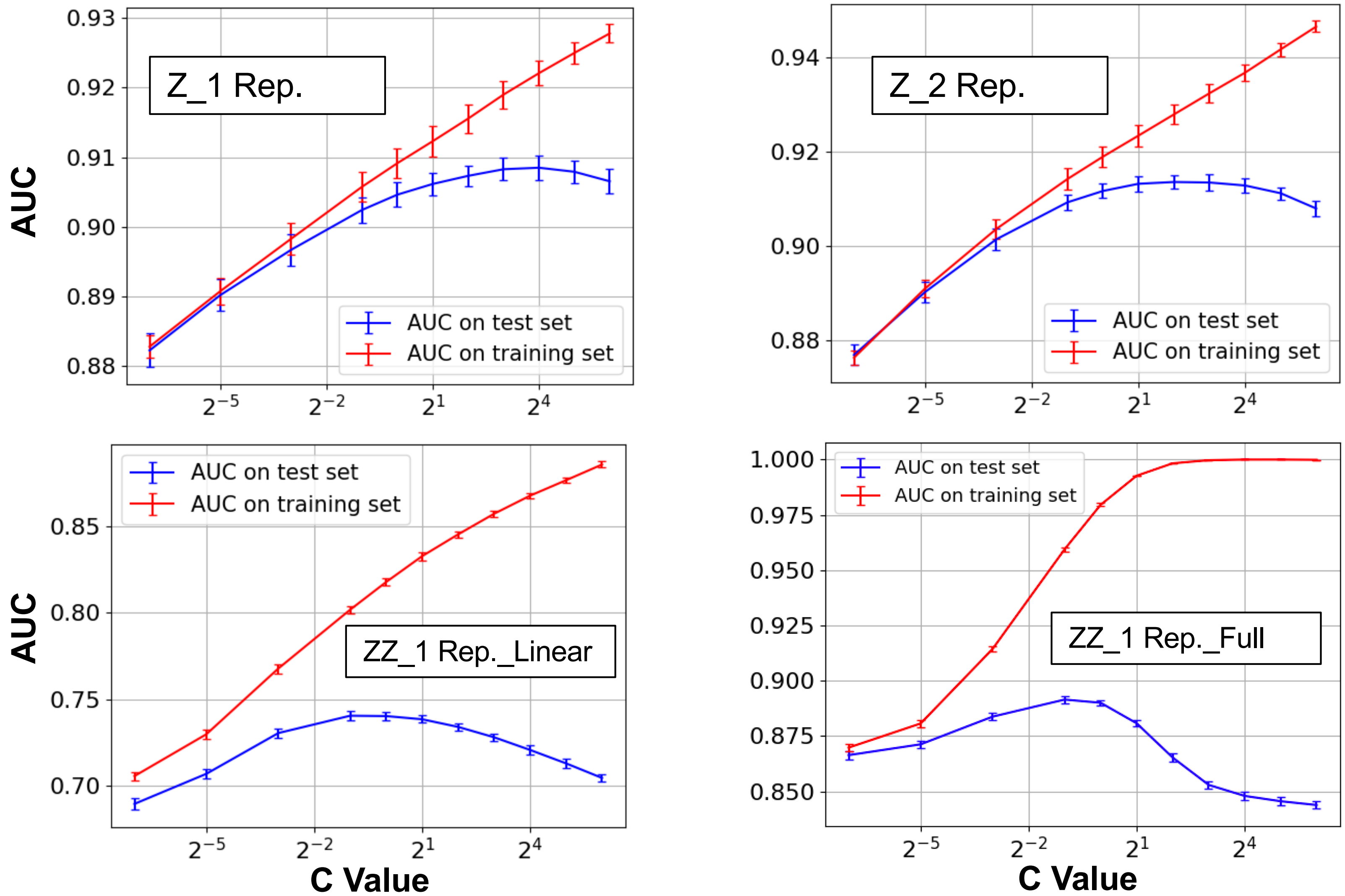
- Various types of encoding circuits are simulated using qiskit
  - A few simple circuits show comparable performance
  - Complicated circuits are prone to overfitting

Classical Model	AUC
SVM	0.91234±0.0030
BDT	0.91292±0.0024
MLP neural network	0.90651±0.0058

Circuit	Rep.	Entanglement	Test set AUC	Training set AUC
X	2	none	0.90834±0.0030	0.91658±0.0021
	3		0.91238±0.0036	0.93055±0.0027
Z	1	none	0.90834±0.0030	0.91658±0.0020
	2		0.91238±0.0036	0.93055±0.0027
	3		0.89240±0.0036	0.90949±0.0009
XX	2	linear	0.73146±0.0037	0.84744±0.0017
		full	0.86332±0.0052	0.99887±0.0001
	3	linear	0.73198±0.0048	0.93766±0.0009
		full	0.72999±0.0047	0.99970±0.0002
ZZ	1	linear	0.73146±0.0037	0.84744±0.0025
		full	0.86332±0.0052	0.99887±0.0002
	2	linear	0.73198±0.0048	0.93767±0.0009
		full	0.72999±0.0047	0.99970±0.0002
	3	linear	0.67960±0.0040	0.88481±0.0004
		full	0.62707±0.0035	0.99964±0.0001
X, XX	2	linear	0.87201±0.0043	0.97902±0.0012
		full	0.88359±0.0021	0.99974±0.0001
	3	linear	0.84779±0.0010	0.99364±0.0002
		full	0.80892±0.0020	1.00000±0.0000
X, ZZ	1	linear	0.73145±0.0037	0.84745±0.0034
		full	0.86332±0.0052	0.99886± 0.0011
	2	linear	0.84441±0.0029	0.99136±0.0013
		full	0.82086±0.0029	1.00000±0.0000
	3	linear	0.84892±0.0029	0.99551±0.0002
		full	0.70668±0.0031	1.00000±0.0000
Z, ZZ	1	linear	0.87423±0.0024	0.97972±0.0004
		full	0.88378±0.0039	0.99974±0.0001
	2	linear	0.84675±0.0022	0.99253±0.0010
		full	0.80875±0.0032	1.00000±0.0000
	3	linear	0.83464±0.0030	0.99512±0.0003
		full	0.69984±0.0026	1.00000±0.0000

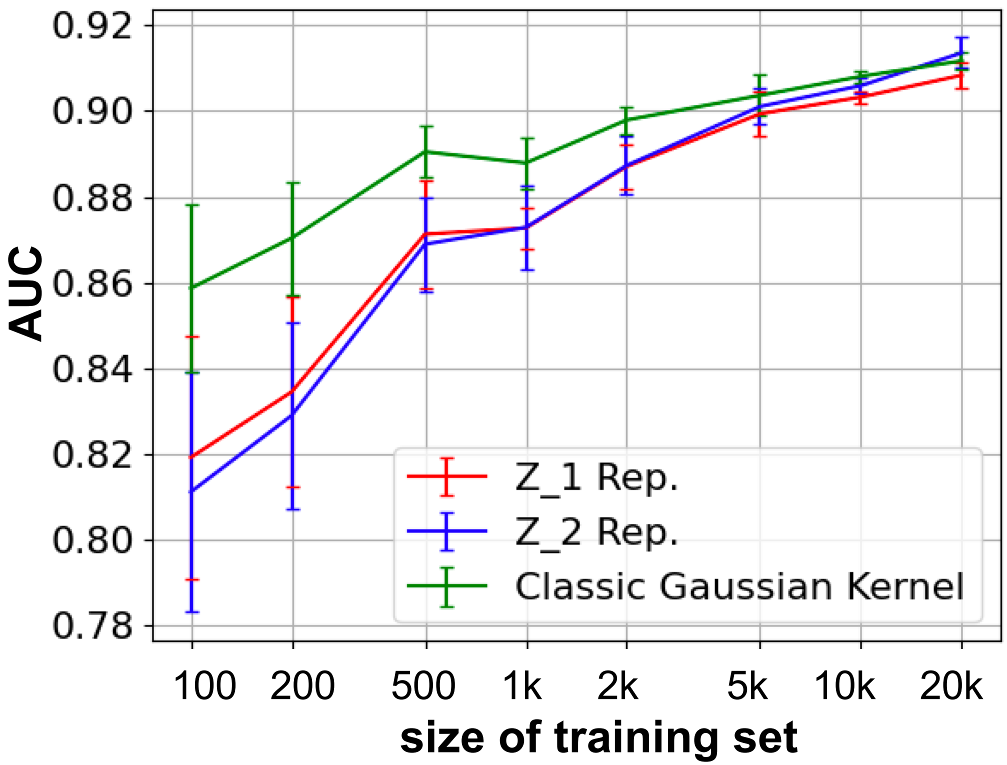
Influence of the Regularization Parameter

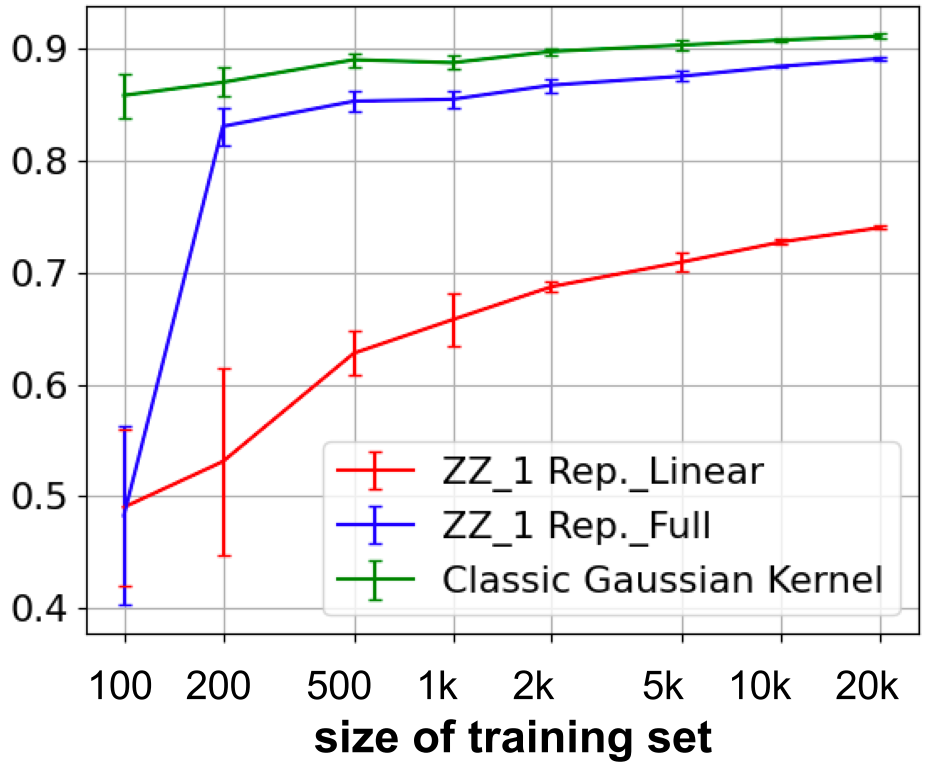
- The influence of the SVM regularization parameter can be carefully tuned to handle the overfitting/underfitting trade-off



Influence of the training size

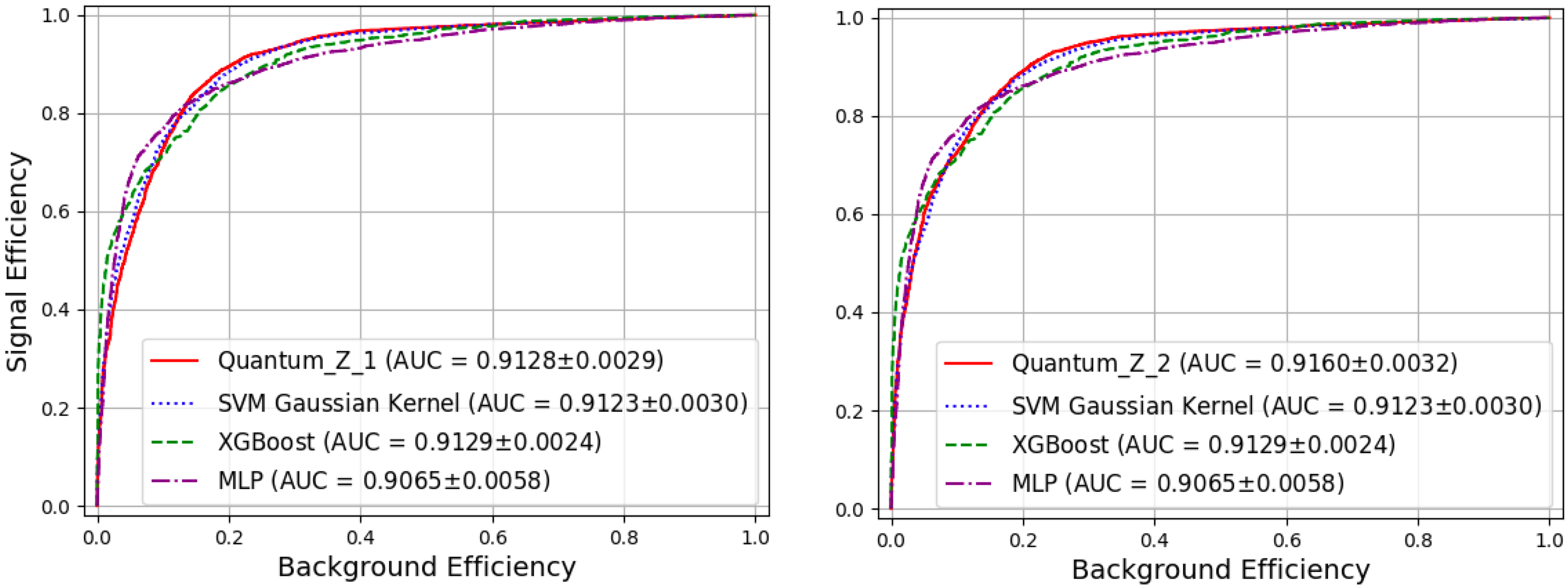
- Different size of the training set are tested
  - The quantum SVM usually shows unstable performance when the training size is small
  - Some circuits start to overtake Gaussian kernel with larger training sets





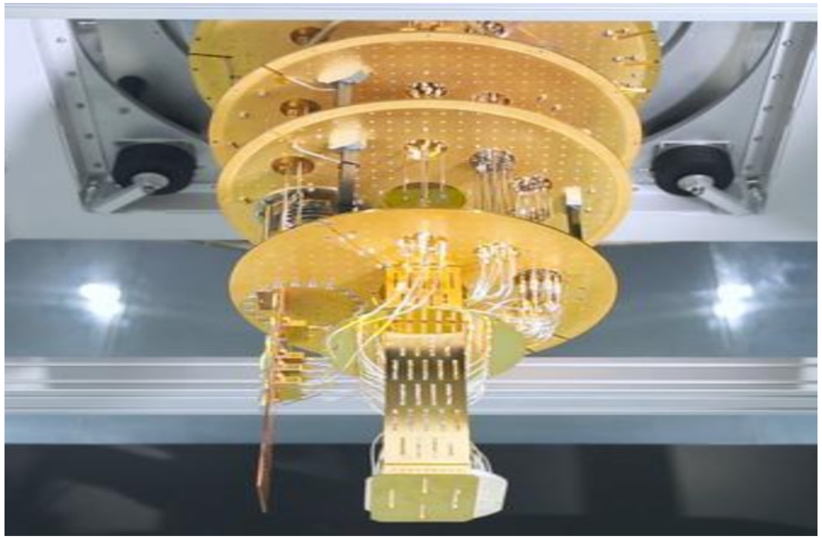
Comparison with Traditional Models

- The discrimination power is compared with the baseline models
  - After the fine tuning of hyper-parameters
  - Similar discrimination power can be achieved

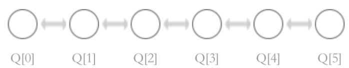


Run on the Quantum Hardware

- It's interesting to see how the noise from real hardware affects the performance
- The OriginQ Wuyuan system based at Hefei, China [8]
  - Based on super-conducting technology
  - 6 qubits, controlled by QPanda API
- Procedure of running QSVM model
  - Design quantum circuits
  - Generate Qpanda code
  - Submit jobs to calculate the Kernel Matrix
  - Train and evaluate the models



本源悟源1号参数介绍

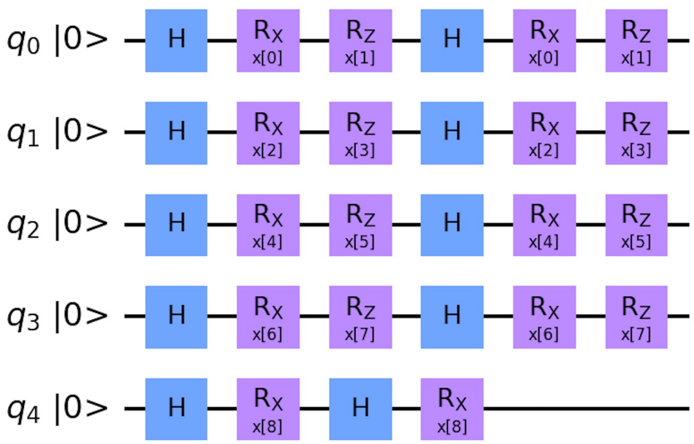


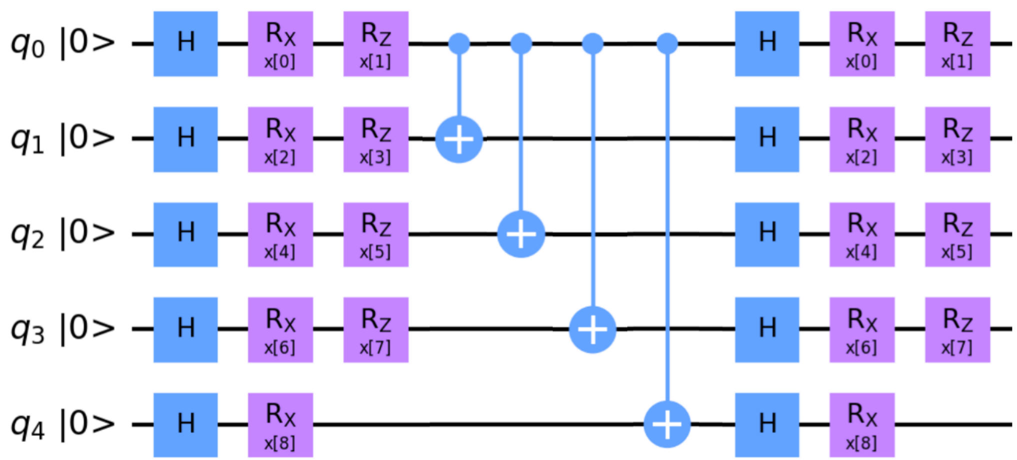
芯片详细参数

量子比特编号	工作频率(MHz)	T1(μs)/T2(μs)	读取保真度F0/F1	平均单比特门 保真度
Q0	5442	17/12.6	0.989/0.965	0.9993
Q1	4470	30/2.3	0.95/0.859	0.9990
Q2	5319	20/2.6	0.975/0.951	0.9990
Q3	4696	32/6.6	0.958/0.923	0.9991
Q4	5214.995	36/3.3	0.984/0.967	0.9992
Q5	4579.685	28/5.4	0.914/0.845	0.9992

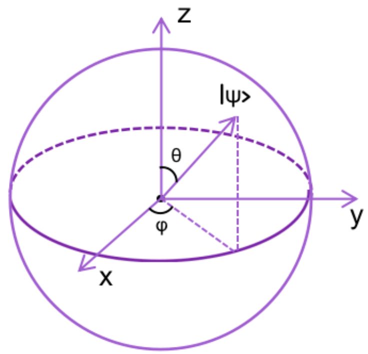
Compressed Feature Map on Quantum Hardware

- Two feature maps are re-designed to meet the limited number of qubits on the Wuyuan system
  - Two features are encoded into each qubit, based on the R_X and R_Z rotations



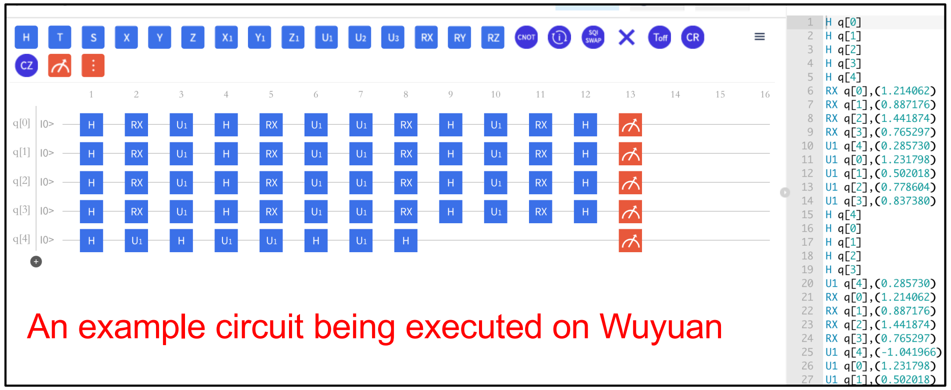


  - The feature map structure is carefully tuned for the best simulation results
    - AUC (1): 0.90373±0.0024
    - AUC (2): 0.91029±0.0023

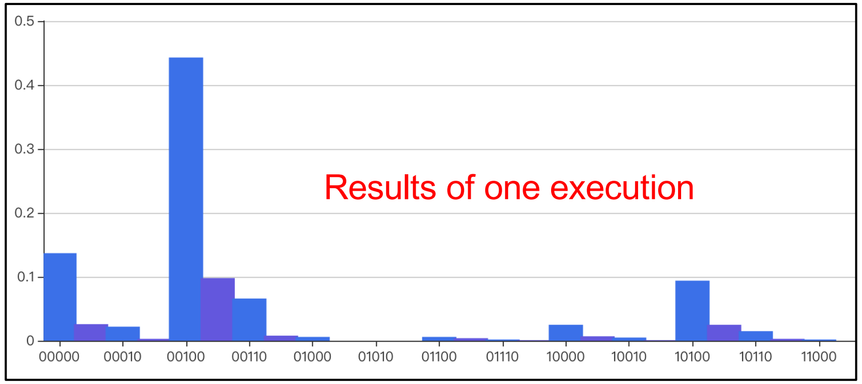


Job Execution on the Wuyuan System

- The quantum circuits are generated based on the dataset, then uploaded to the Wuyuan system via QPanda
  - Quantum circuits are automatically optimized against the Wuyuan system

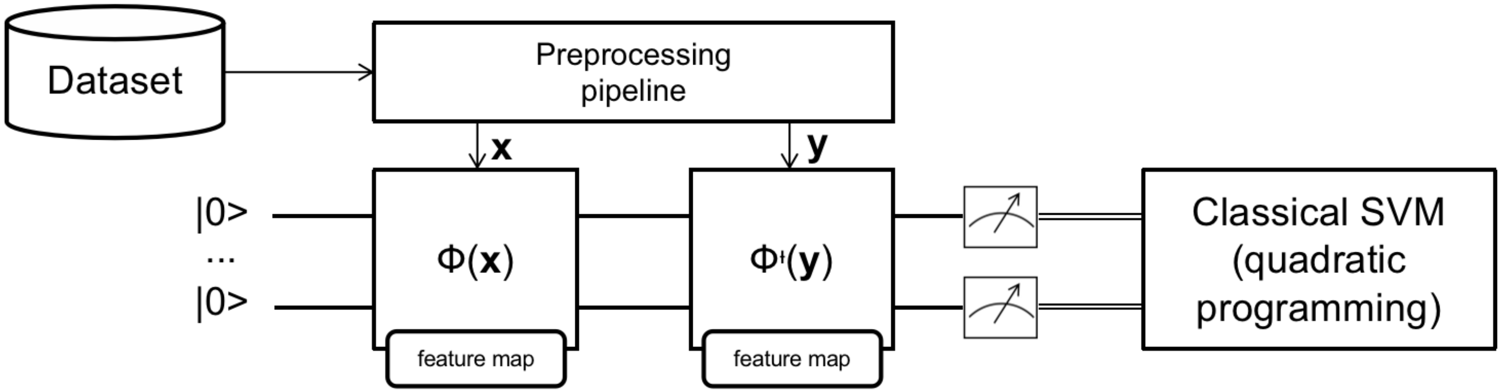


An example circuit being executed on Wuyuan



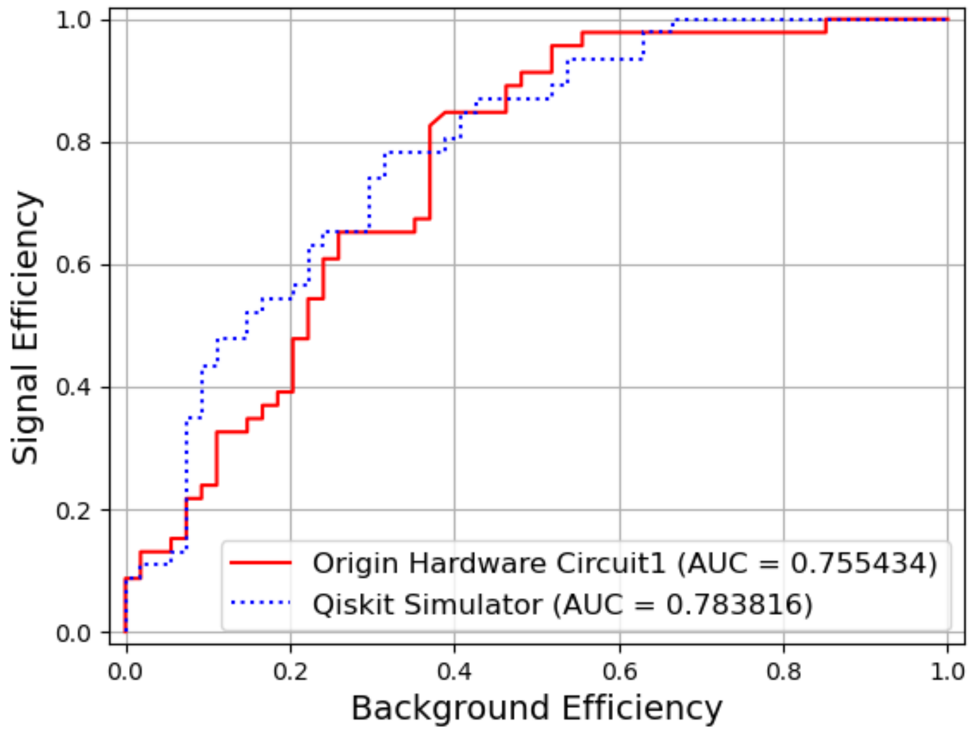
Results of one execution

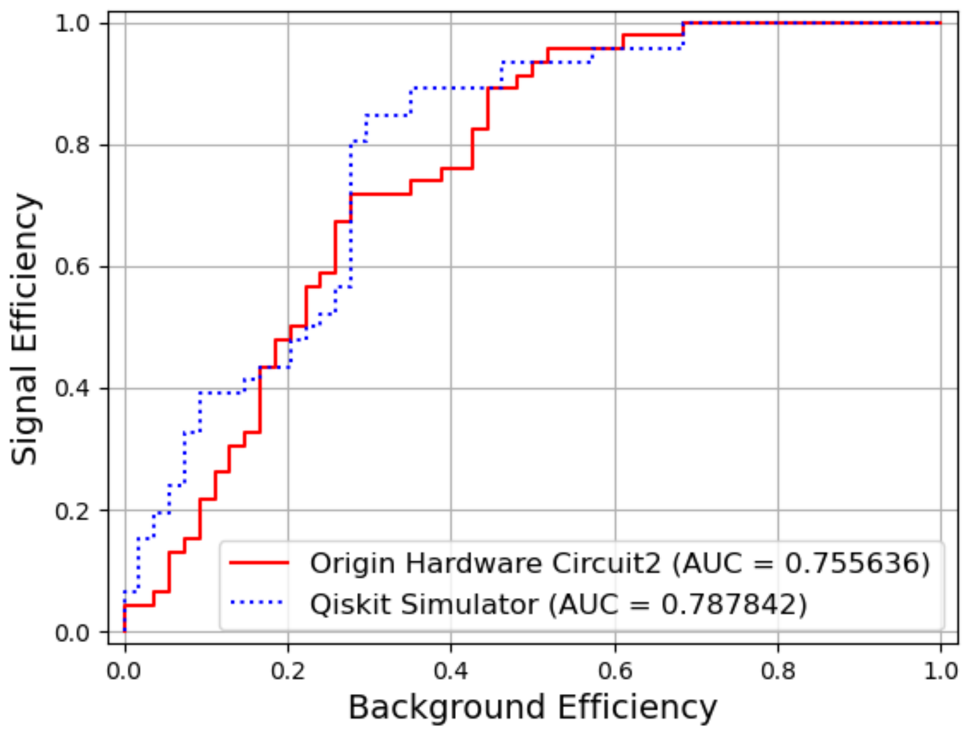
  - Results are transferred back to the classical computer for downstream computations



Results from the Hardware

- Results from Origin Wuyuan
  - Results obtained from 100 training tracks and 100 test tracks, averaged from three runs
  - The noise compromises the performance, but at a controllable level

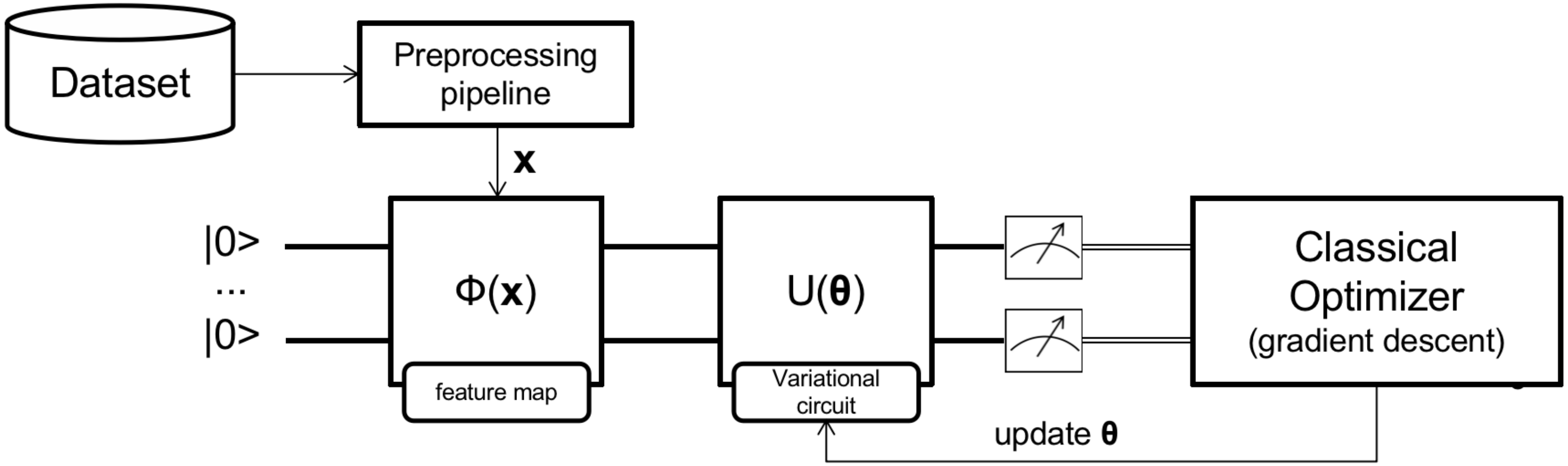




Method Ⅱ
Variational Quantum Classifier

Variational Quantum Classifier: Introduction

❖ Variational Quantum Classifier as a hybrid model



- A subsequent variational (train-able) circuit performs a linear transformation on the prepared state
- The parameters of the variational circuit can be trained based on the gradients calculated classically
- Data is classified by measuring the output qubit(s). (estimating the probabilities of each state)

Optimization of VQC

- As a 'quantum neural network', the key issue is to optimize the free parameters of the variational circuits
  - Traditional backward propagation is inpractical due to the limits of quantum theory
  - The numerical differentiation method was usually used previously to calculate the gradients

$$\frac{\mathrm{d}f(\theta)}{\mathrm{d}\theta} \approx \frac{f(\theta+h)-f(\theta-h)}{2h} + \boxed{O\left(h^2\right)}$$

the *error* term

  - Currently, the gradient is more popularly computed based on the **parameter shift rule** [9]

$$\frac{d}{d\theta} f(\theta) = r(f(\mu+s) - f(\mu-s))$$

Search of Optimal Encoding Circuits

- A wide range of encoding circuits is simulated as well
  - Relatively simpler (X_3, Z_2) circuits provide better discrimination power. This is simillar with results from qSVM
  - Overfitting is less obvious comparing to qSVM

Circuit	Repetitions[1]	Entanglement[2]	Training set AUC[3]	Test set AUC[3]
X	2	none	0.85733±0.0158	0.83719±0.0334
	3		0.86581±0.0242	0.84354±0.0309
XX	2	linear	0.70366±0.2394	0.62051±0.0511
		full	0.74656±0.1962	0.61870±0.0456
	3	linear	0.69882±0.0331	0.56579±0.0850
		full	0.73998±0.0367	0.53483±0.0422
Z	1	none	0.84914±0.0416	0.81800±0.0607
	2		0.87338±0.0134	0.84729±0.0282
	3		0.75637±0.0349	0.70577±0.0965
ZZ	1	linear	0.69493±0.0274	0.59169±0.0770
		full	0.76208±0.2652	0.61317±0.0334
	2	linear	0.71653±0.0280	0.61720±0.0519
		full	0.72435±0.0267	0.55957±0.0497
	3	linear	0.71762±0.0274	0.59083±0.0377
		full	0.75620±0.0242	0.50290±0.0308
X,XX	2	linear	0.68095±0.0195	0.48709±0.0310
		full	0.74749±0.0258	0.52300±0.0423
	3	linear	0.68118±0.0165	0.51419±0.0288
		full	0.75874±0.0262	0.52024±0.0295
Z,ZZ	1	linear	0.65018±0.0100	0.49202±0.0327
		full	0.75312±0.0161	0.50179±0.0274
	2	linear	0.67567±0.0243	0.50894±0.0372
		full	0.75500±0.0259	0.50336±0.0324
	3	linear	0.65160±0.0179	0.50692±0.0387
		full	0.75223±0.0195	0.50762±0.0359
X,ZZ	1	linear	0.66164±0.0242	0.49825±0.0429
		full	0.74262±0.0202	0.50841±0.0288
	2	linear	0.66667±0.0248	0.50124±0.0381
		full	0.75893±0.0163	0.49098±0.0483
	3	linear	0.68708±0.0228	0.50050±0.0531
		full	0.75371±0.0240	0.48624±0.5777

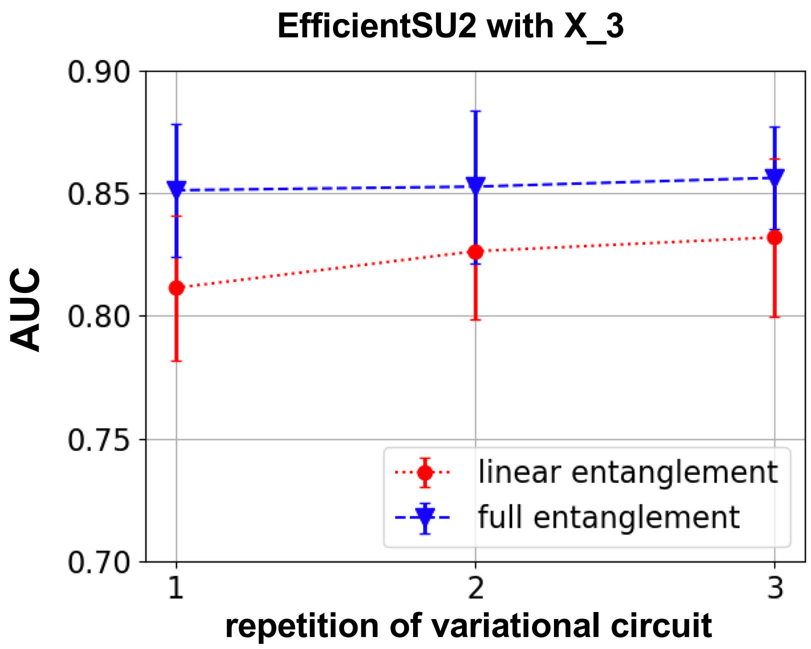
Search of Optimal Variational Circuits

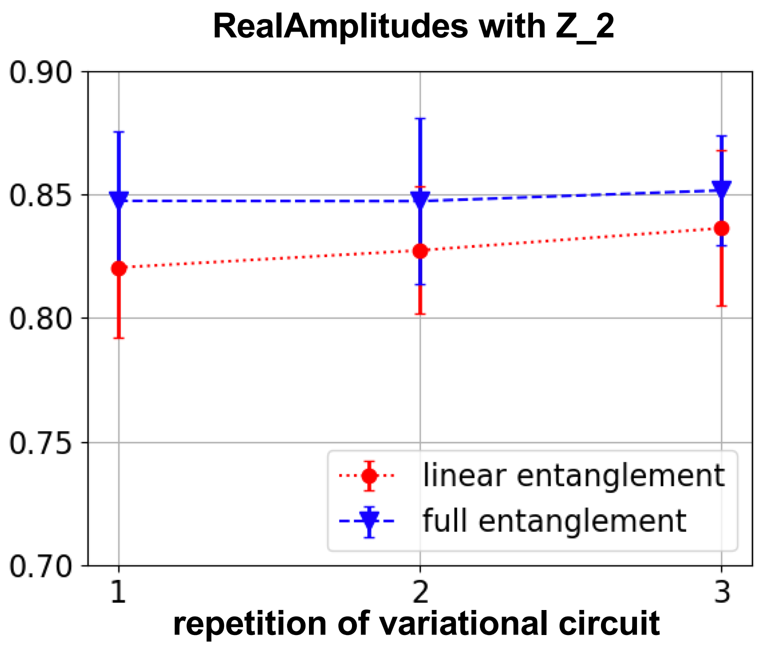
- A set of pre-studied N-Local ansatz are scanned based on the optimal encoding circuits
  - EfficientSU2: ansatz with single qubit spanned by SU(2) and CX
  - PauliTwoDesign: ansatz with single qubit Pauli rotations and pairwise CZ entanglements
  - RealAmplitudes: ansatz with single qubit Y rotations and pairwise CX entanglements
  - TwoLocal: ansatz with flexible rotation layers and entanglement layers
  - ExcitationPreserving: heuristic excitation-preserving wave function ansatz
- For X_3 and Z_2, the best variational circuits are EfficientSU2 and RealAmplitudes, respectively

variational circuit	Test set AUC with X_3	Test set AUC with Z_2
EfficientSU2	0.84983±0.0292	0.84514±0.0365
ExcitationPreserving	0.42660±0.0327	0.52333±0.0280
PauliTwoDesign	0.74614±0.0277	0.76688±0.0425
RealAmplitudes	0.84656±0.0303	0.84711±0.0277
TwoLocal	0.84102±0.0330	0.84707±0.0272

Search of Optimal Variational Cirtcuits

- Performance of different ansatz structures
  - Different entanglement methods and ansatz depth are simulated to study the impact on the performance
  - In general, the full entanglement method, and deeper ansatz (with more trainable parameters) always gives better discrimination power, but also consumes much more computing resource

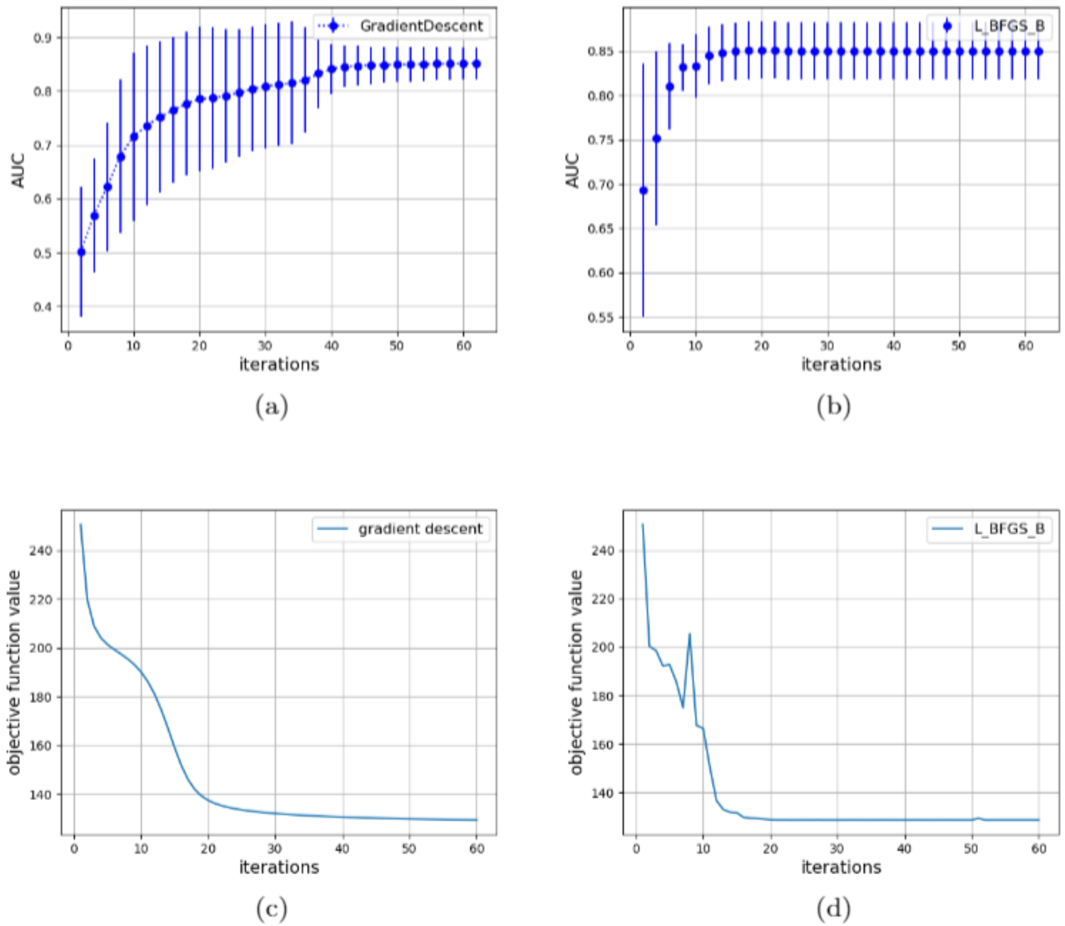




Optimization of VQC

❖ Common gradient descent method and Quasi-Newton method are compared

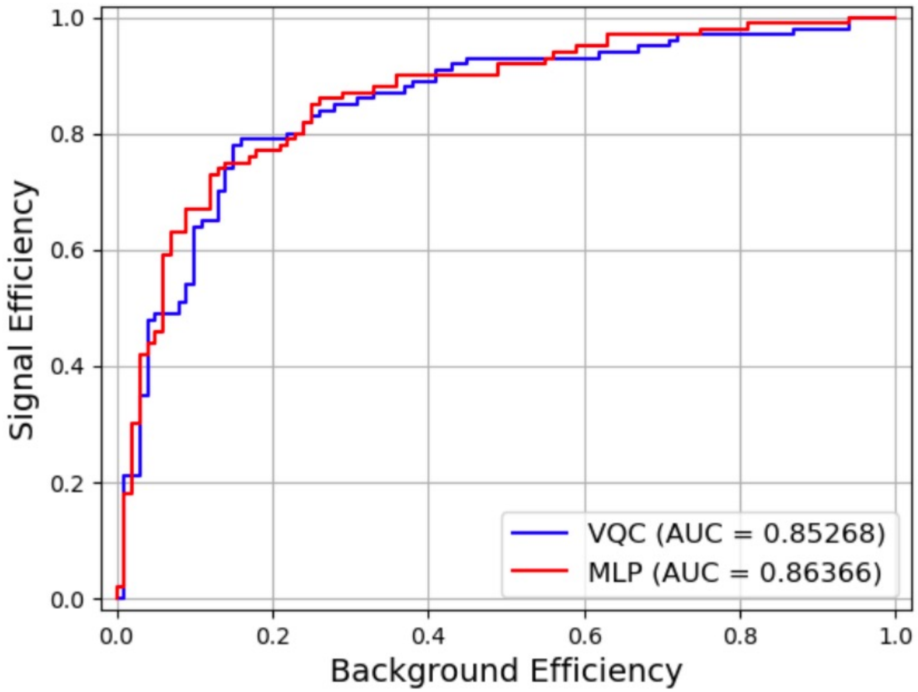
- Since L_BFGS_B (Quasi-Newton method) invokes the second derivative of the loss function, the convergence can be achieved much faster



(a)(c) Variation of AUC with the number of iterations
(b)(d) Variation in the objective function during the iteration.

Comparison with Classic MLP

- The optimal VQC is compared with the classical MLP neural network
  - VQC config: EfficientSU2 with X_3 and L_BFGS_B optimizer
  - MLP config: 400x200x100x50x15 (relu, adam)
- On small samples, VQC performs similarly to the classical MLP neural network



Summary (1)

- Quantum Computing has made rapid progress and has the potential to revolutionise science and society in the next five to ten years.
- Quantum machine learning could possibly become a valuable alternative to classical machine learning for HEP data processing and physics analysis
- An International Collaboration (CERN QTI) serves as an international and open platform to build collaborations and define the roadmap and research programme
- Several HEP groups have been formed to push the application of QML in China, but we need more efforts, more collaborations and more …

Summary (2)

- Targeting at the BESIII μ/π identification problem, we studied the qSVM and VQC algorithms as a proof of concept
  - A wide range of encoding methods are evaluated
    - A few ones show comparable performance with classical models
    - Others show potential to classify much more complicated data
  - Efforts are made to run the qSVM model on the Wuyuan system
  - Different optimization method and variation ansatzs are studied for the VQC
    - The design of ansatzs heavily depends on the specific problem
    - Automated way to find optimal ansatzs is desired
- The QML models show quite comparable performance comparing to their classical counterparts, showing potential to apply QML on HEP experiments

Thanks for your attention!

References

[1] Quantum support vector machine for big data classification, Phys. Rev. Lett. 113, 130503 (2014)

[2] Effect of data encoding on the expressive power of variational quantum-machine-learning models, Phys. Rev. A 103, 032430 (2021)

[3] Application of quantum machine learning using the quantum kernel algorithm on high energy physics analysis at the lhc (2021). arXiv:2104.05059

[4] Higgs analysis with quantum classifiers, CHEP2021

[5] Quantum machine learning for particle physics using a variational quantum classifier, JHEP 2021 (2)

[6] Quantum Machine Learning in Feature Hilbert Spaces, Phys. Rev. Lett. 122, 040504 (2019)

[7] Supervised learning with quantum-enhanced feature spaces, Nature volume 567, pages 209–212 (2019)

[8] https://qcloud.originqc.com.cn/computing (2021.11)

[9] Evaluating analytic gradients on quantum hardware. Phys. Rev. A, 99(3):032331, 2019

QML Tutorial

Teng Li

Tutorial: Setup Qiskit

- Qiskit is an open-source SDK for working with quantum computers at the level of pulses, circuits, and application modules.
- qSVM is builtin within Qiskit
- Setup Qiskit via Anaconda and pip

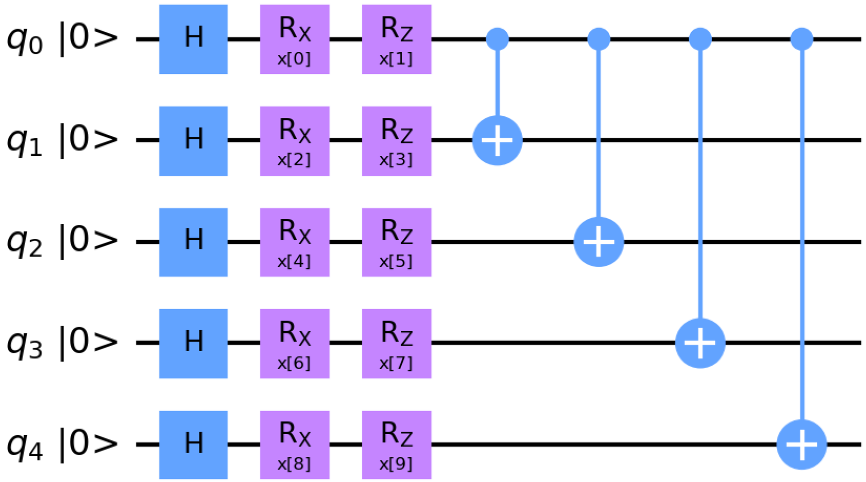
```
$ conda create -n qml python=3
$ conda activate qml
$ pip install 'qiskit[visualization]'
$ pip install qiskit_machine_learning
$ pip install qiskit-aer-gpu
```

Build qSVM from scratch

Tutorial: Building qSVM from scratch

- Build and visualize a encoding circuit via QuantumCircuit

```python
from qiskit.circuit import QuantumCircuit, ParameterVector

if __name__ == "__main__":

  num_qubits = 5   # Number of qubits used by the feature map
  feature_map = QuantumCircuit(num_qubits)

  # Design the structure of the feature map
  x = ParameterVector('x', length=10) # Parameters associate with the feature map
  for i in range(5):
    feature_map.h(i)                  # Hadmard gate
    feature_map.rx(x[i*2], i)         # RX gate
    feature_map.rz(x[i*2+1], i)       # RZ gate

  # Entanglement
  for j in range(1,5):
    feature_map.cx(0,j)
  # Draw the circuits
  feature_map.draw(output='mpl', filename='circuit.png', initial_state=True)
```



You can also use an online editor to design your circuit, then implement it in python
https://qcloud.originqc.com.cn/quantumVm/5/0

Draw the circuit to visualize and validate it.

Tutorial: Building qSVM from scratch

- Build a quantum kernel generator using this feature map

```
from qiskit.circuit import QuantumCircuit, ParameterVector
from qiskit import Aer
from qiskit.utils import QuantumInstance
from qiskit_machine_learning.kernels import QuantumKernel

if __name__ == "__main__":

  num_qubits = 5   # Number of qubits used by the feature map
  feature_map = QuantumCircuit(num_qubits)

  # Design the structure of the feature map
  x = ParameterVector('x', length=10) # Parameters associate with the feature map
  for i in range(5):
    feature_map.h(i)                  # Hadmard gate
    feature_map.rx(x[i*2], i)         # RX gate
    feature_map.rz(x[i*2+1], i)       # RZ gate

  # Entanglement
  for j in range(1,5):
    feature_map.cx(0,j)

  # Create a quantum instance (aer simulator)
  q_backend = QuantumInstance(Aer.get_backend('aer_simulator_statevector'),shots=8000,
                              seed_simulator=42,seed_transpiler=42)
  # Associate feature map with the Quantum Kernel
  q_kernel = QuantumKernel(feature_map=feature_map, quantum_instance=q_backend)
```

QuantumKernel will build circuit based on your feature map.

The q_kernel.evaluate will calculate the kernel based on the input data

Tutorial: Building qSVM from scratch

- Train and evaluate SVM with the quantum kernel

```python
from qiskit.circuit import QuantumCircuit, ParameterVector
from qiskit import Aer
from qiskit.utils import QuantumInstance
from qiskit_machine_learning.kernels import QuantumKernel
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
import pandas as pd

if __name__ == "__main__":

  num_qubits = 5   # Number of qubits used by the feature map
  feature_map = QuantumCircuit(num_qubits)

  # Design the structure of the feature map
  x = ParameterVector('x', length=10) # Parameters associate with the feature map
  for i in range(5):
    feature_map.h(i)                 # Hadmard gate
    feature_map.rx(x[i*2], i)        # RX gate
    feature_map.rz(x[i*2+1], i)      # RZ gate

  # Entanglement
  for j in range(1,5):
    feature_map.cx(0,j)

  # Create a quantum instance (aer simulator)
  q_backend = QuantumInstance(Aer.get_backend('aer_simulator_statevector'),shots=8000,
                              seed_simulator=42,seed_transpiler=42)
  # Associate feature map with the Quantum Kernel
  q_kernel = QuantumKernel(feature_map=feature_map, quantum_instance=q_backend)

  # Build SVM with the Quantum Kernel
  svc = SVC(kernel=q_kernel.evaluate, C=5)

  # Evaluate the model
  train_data = pd.read_csv("data.csv")
  train_data_tag = train.pop('tag')
  scores = cross_val_score(svc, train_data, train_data_tag, cv=3)
  print(scores.mean())
  print(scores.std())
```

Just use sklearn.SVC to create a SVM model, with a self-defined kernel

Tutorial: Building qSVM from scratch

- Use built-in feature maps in Qiskit

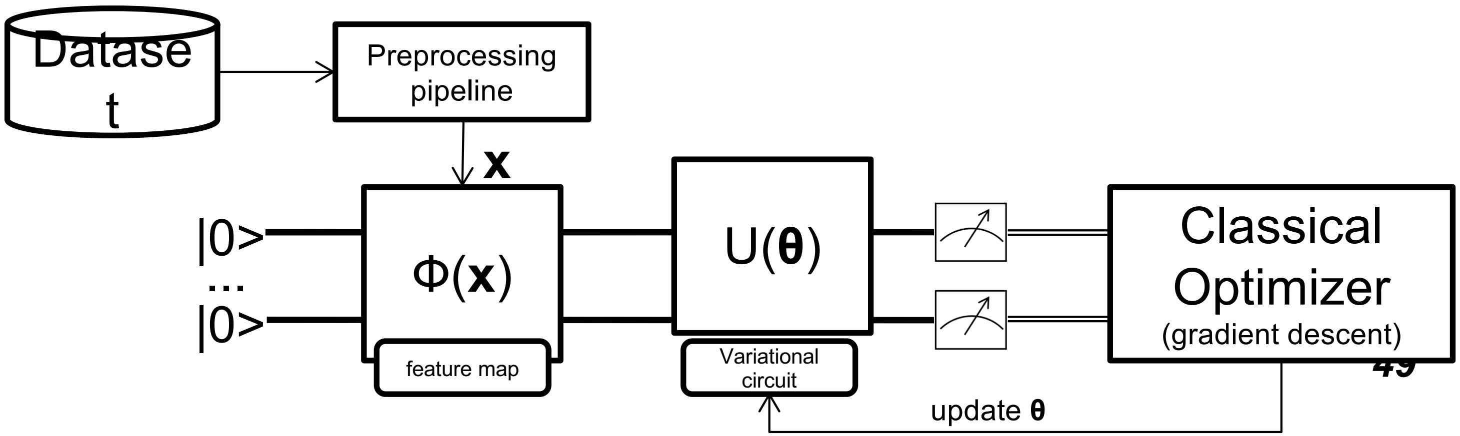
```
from qiskit.circuit.library import PauliFeatureMap
from qiskit import Aer
from qiskit.utils import QuantumInstance
from qiskit_machine_learning.kernels import QuantumKernel
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
import pandas as pd

if __name__ == "__main__":


  feature_map = PauliFeatureMap(feature_dimension=10,reps=2, paulis=['Z','ZZ'], entanglement='full')

  # Create a quantum instance (aer simulator)
  q_backend = QuantumInstance(Aer.get_backend('aer_simulator_statevector'),shots=8000,
                              seed_simulator=42,seed_transpiler=42)
  # Associate feature map with the Quantum Kernel
  q_kernel = QuantumKernel(feature_map=feature_map, quantum_instance=q_backend)

  # Build SVM with the Quantum Kernel
  svc = SVC(kernel=q_kernel.evaluate, C=5)

  # Evaluate the model
  train_data = pd.read_csv("data.csv")
  train_data_tag = train.pop('tag')
  scores = cross_val_score(svc, train_data, train_data_tag, cv=3)
  print(scores.mean())
  print(scores.std())
```

Checkout the built-in feature maps in Qiskit:
https://qiskit.org/documentation/apidoc/circuit_library.html#data-encoding-circuits

Build VQC from scratch

Introduction

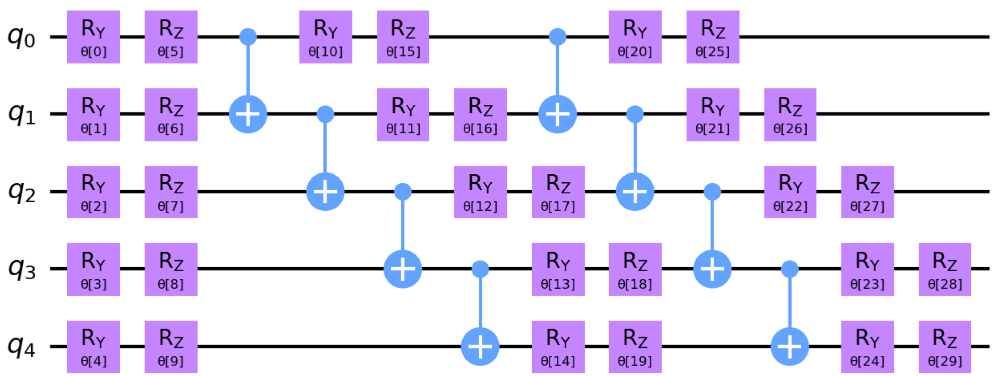
- Besides the feature map, VQC uses an additional variational circuit to transform the input state
- The variational circuit is trainable via gradient descent



Tutorial: Building VQC from scratch

❖ Create the ansatz of the variational circuit

```python
from qiskit.circuit.library import PauliFeatureMap,EfficientSU2
from qiskit import Aer
from qiskit.utils import QuantumInstance
import pandas as pd

if __name__ == "__main__":

  var_ansatz = EfficientSU2(5, entanglement='linear', reps=2)
  var_ansatz.draw(output='mpl', filename='ansatz.png')
```



Like the feature map, you can also use the built-in, or self-defined ansatz structure.

Then visualize the structure of the ansatz.
(θ[n] is the trainable parameters)

Checkout the built-in ansatz in Qiskit:
https://qiskit.org/documentation/apidoc/circuit_library.html#n-local-circuits

Tutorial: Building VQC from scratch

- Built VQC based on a feature map and a variational circuit

```python
from qiskit.circuit.library import PauliFeatureMap,EfficientSU2
from qiskit import Aer
from qiskit.utils import QuantumInstance
from qiskit_machine_learning.algorithms import VQC
from qiskit.algorithms.optimizers import GradientDescent
import pandas as pd

if __name__ == "__main__":

  feature_map = PauliFeatureMap(feature_dimension=5,reps=2, paulis=['Z','ZZ'], entanglement='full')
  var_ansatz = EfficientSU2(5, entanglement='linear', reps=2)

  ## Create the GradientDescent optimizer
  optimizer = GradientDescent(maxiter=100, learning_rate=0.01)

  q_backend = QuantumInstance(Aer.get_backend('aer_simulator_statevector'), shots=1024,
                              seed_simulator=42, seed_transpiler=42)
  # Create the VQC, based on the feature map and variational circuit
  vqc = VQC(num_qubits=5, feature_map=feature_map, ansatz=var_ansatz, loss='cross_entropy',
            optimizer=optimizer, quantum_instance=q_backend)

  # Fit VQC
  train_data = pd.read_csv("data.csv")
  train_data_tag = train_data.pop("tag")
  vqc.fit(train_data.to_numpy(),train_data_tag.to_numpy())
```

Tutorial: Building VQC from scratch

- The key issue for VQC is the optimization problem

- There are a lot of built-in optimizers in Qiskit:
  - https://qiskit.org/documentation/stubs/qiskit.algorithms.optimizers.html

- You could also develop your own optimizer if you are familiar with quantum algorithms
  - Check https://pennylane.ai/qml/demos_optimization.html for more technical details

Welcome to collaborate together on developing Quantum Algorithms and their applications in HEP data processing and analysis.