Quantum Machine Learning in HEP with `Qibo`

PyHEP 2024

Matteo Robbiati[†] on behalf of the Qibo team[‡]

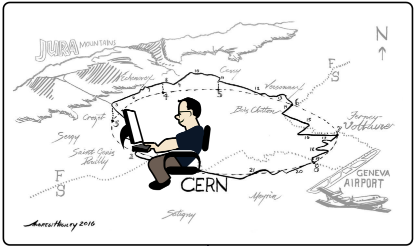
3 July 2023

[†] PhD candidate, University of Milan, Italy and CERN, Switzerland.
✉ `matteo.robbiati@cern.ch`

[‡] *https : //qibo.science/*

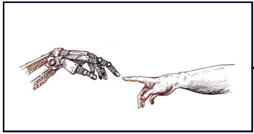
Quantum Computing for HEP

High Energy Physics

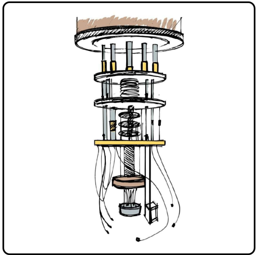


Plenty of data to analyze

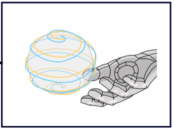
Need for technological development



Quantum computers



Computational boost (problem specific)



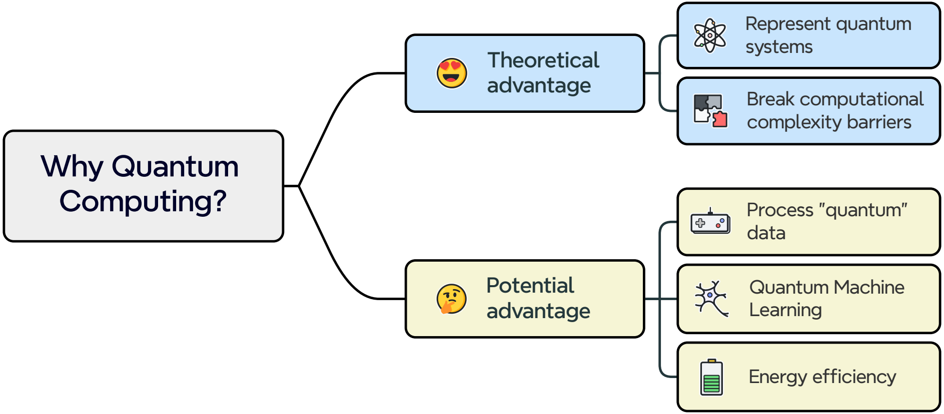
“ For HEP! ”

Noise and limited resources

- Simulation
- Control
- Calibration

Quantum Computing in a nutshell

A quite general "why?"

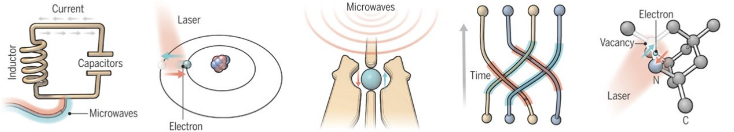


Why Quantum Computing?

- Theoretical advantage
  - Represent quantum systems
  - Break computational complexity barriers
- Potential advantage
  - Process "quantum" data
  - Quantum Machine Learning
  - Energy efficiency

Gate based quantum computing

1. classical bits are replaced by **qubits** $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ (quantum states).



Superconducting loops
A resistance-free current oscillates back and forth around a circuit loop. An injected microwave signal excites the current into superposition states.

Trapped ions
Electrically charged atoms, or ions, have quantum energies that depend on the location of electrons. Tuned lasers cool and trap the ions, and put them in superposition states.

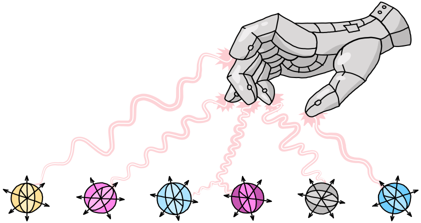
Silicon quantum dots
These “artificial atoms” are made by adding an electron to a small piece of pure silicon. Microwaves control the electron’s quantum state.

Topological qubits
Quasiparticles can be seen in the behavior of electrons channeled through semiconductor structures. Their braided paths can encode quantum information.

Diamond vacancies
A nitrogen atom and a vacancy add an electron to a diamond lattice. Its quantum spin state, along with those of nearby carbon nuclei, can be controlled with light.

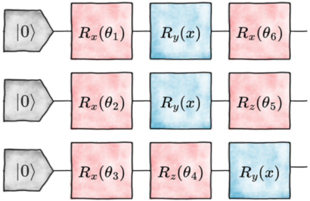
Gate based quantum computing

1. classical bits are replaced by **qubits** $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ (quantum states).
2. we can manipulate the qubit state applying **gates**: $|\psi'\rangle = \mathcal{U}(\boldsymbol{\theta})\,|\psi\rangle$.
   Typically we use 1-qubit and 2-qubits gates!



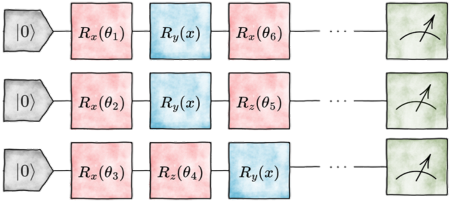
Gate based quantum computing

1. classical bits are replaced by **qubits** $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ (quantum states).
2. we can manipulate the qubit state applying **gates**: $|\psi'\rangle = \mathcal{U}(\boldsymbol{\theta})|\psi\rangle$.
   Typically we use 1-qubit and 2-qubits gates!
3. combine together gates to build **quantum circuits**;



Gate based quantum computing

1. classical bits are replaced by **qubits** $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ (quantum states).
2. we can manipulate the qubit state applying **gates**: $|\psi'\rangle = \mathcal{U}(\boldsymbol{\theta})|\psi\rangle$.
   Typically we use 1-qubit and 2-qubits gates!
3. combine together gates to build **quantum circuits**;
4. to access the information we need to measure the system.



>_ Example 1: preparing entangled states

With quantum computing, we introduce new tools.

- prepare a quantum state in the computational zero $|0\rangle$;
- we can prepare superposition:

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \quad \text{with} \quad H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix},\ |0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix},\ |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix};$$

- let's apply a controlled-NOT (CNOT) gate on a second qubit prepared in $|0\rangle$:

$$\mathrm{CNOT}\left(\underbrace{\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)}_{\text{control}} \otimes |0\rangle\right) = \frac{1}{\sqrt{2}}(|\textcolor{blue}{0}0\rangle + \mathrm{NOT}_{\mathrm{targ}}|\textcolor{red}{1}0\rangle) = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle).$$

>_ Example 1: preparing entangled states

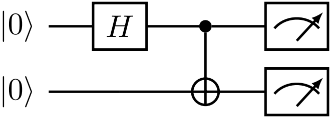
With quantum computing, we introduce new tools.

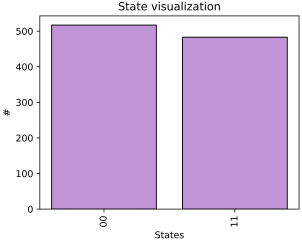
- prepare a quantum state in the computational zero $|0\rangle$;
- we can prepare superposition:

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \quad \text{with} \quad H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \ |0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \ |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix};$$

- let's apply a controlled-NOT (CNOT) gate on a second qubit prepared in $|0\rangle$:

$$\text{CNOT}\left(\underbrace{\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)}_{\text{control}} \otimes |0\rangle\right) = \frac{1}{\sqrt{2}}(|{\color{blue}0}0\rangle + \text{NOT}_{\text{targ}}\,|{\color{red}1}0\rangle) = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle).$$

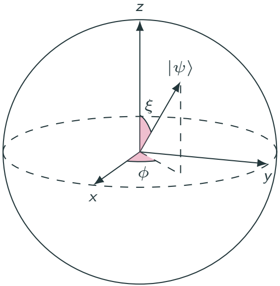


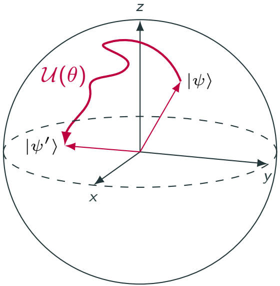


Parametric gates prepare variational quantum states

- Among the gates, parametric ones can be useful!
- Let's consider a single qubit system:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \qquad \text{with} \qquad \alpha = \cos\frac{\theta}{2}, \quad \beta = e^{i\phi}\sin\frac{\theta}{2}.$$



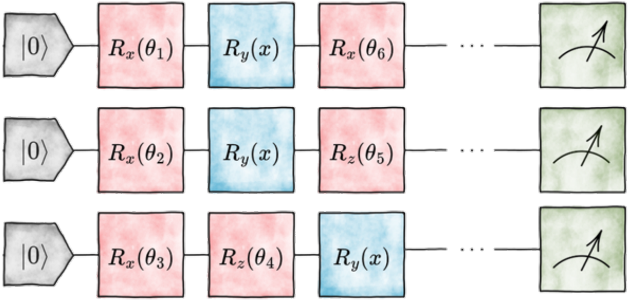


We can use as parametric gates the rotation around the axis of the block sphere:

$$R_k(\theta) = \exp\left[-i\theta\sigma_k\right], \qquad \text{with} \qquad \sigma_k \in \{I, \sigma_x, \sigma_y, \sigma_z\}.$$

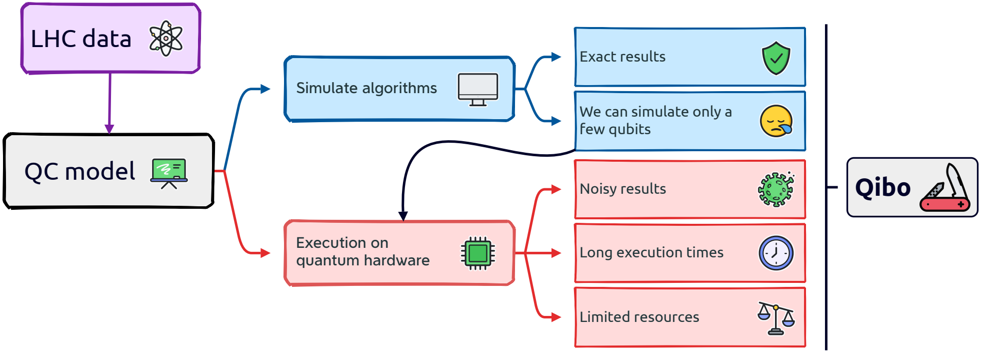
Parametric quantum circuits

Parametric gates can be used to build parametric quantum circuits.



Qibo 0.2.9

What is needed for doing quantum computing?



LHC data

QC model

Simulate algorithms

Execution on quantum hardware

Exact results

We can simulate only a few qubits

Noisy results

Long execution times

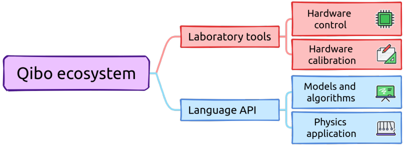
Limited resources

Qibo

The Qibo ecosystem

arXiv:2009.01845

Qibo is an **open-source** hybrid quantum operating system for self-hosted quantum computers.



Qibo ecosystem

Laboratory tools

Hardware control

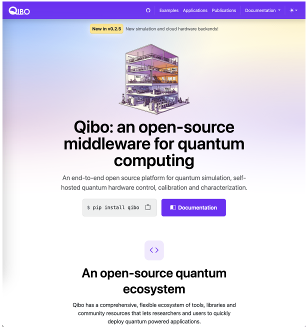
Hardware calibration

Language API

Models and algorithms

Physics application

1. Fully open-source and community driven.
2. Modular layout design with possibility of adding:
    - new backends for simulation,
    - new platforms for hardware control,
    - new drivers for control electronics.
3. Supported by documentation and tests/CI on quantum hardware.



New in v0.2.5 New simulation and cloud hardware backends!

Qibo: an open-source middleware for quantum computing

An end-to-end open source platform for quantum simulation, self-hosted quantum hardware control, calibration and characterization.

```
$ pip install qibo
```

An open-source quantum ecosystem

Qibo has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers and users to quickly deploy quantum powered applications.

https://qibo.science

The Qibo timeline

2019
Design proposal

2020
Language API

2021
Classical simulation

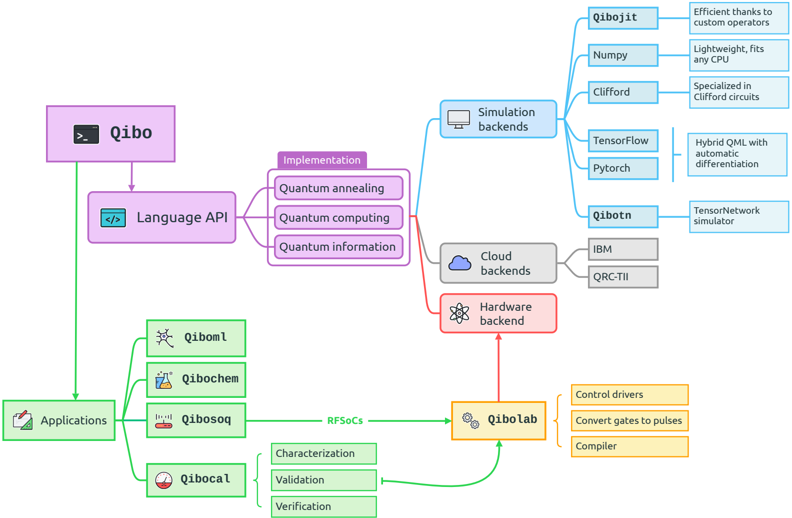
2022
Lab drivers

2023
Quantum control

2024
Calibration and applications

An open source and full-stack framework for QC

arXiv:2308.06313



Qibo

Language API

Implementation

Quantum annealing

Quantum computing

Quantum information

Simulation backends

Qibojit – Efficient thanks to custom operators

Numpy – Lightweight, fits any CPU

Clifford – Specialized in Clifford circuits

TensorFlow

Pytorch

Hybrid QML with automatic differentiation

Qibotn – TensorNetwork simulator

Cloud backends

IBM

QRC-TII

Hardware backend

Applications

Qiboml

Qibochem

Qibosoq

Qibocal

RFSoCs

Qibolab

Control drivers

Convert gates to pulses

Compiler

Characterization

Validation

Verification

https://github.com/qiboteam

A focus on classical simulation performances

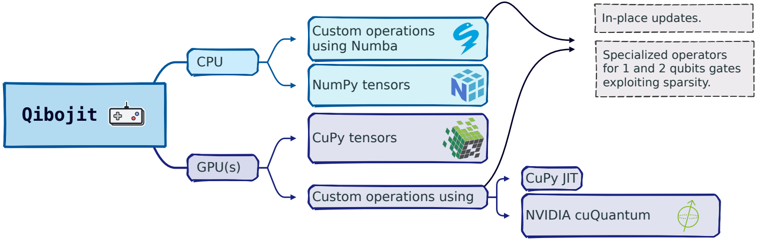
arXiv:2203.08826

State vector simulation solves:

$$\psi'(\sigma_1,\ldots,\sigma_n) = \sum_{\boldsymbol{\tau}'} G(\boldsymbol{\tau},\boldsymbol{\tau}')\psi(\sigma_1,\ldots,\boldsymbol{\tau}',\ldots,\sigma_n)$$

The number of operations scales exponentially with the number of qubits.

Qibo uses just-in-time technology and hardware acceleration:



Qibojit

CPU

Custom operations using Numba

NumPy tensors

GPU(s)

CuPy tensors

Custom operations using

CuPy JIT

NVIDIA cuQuantum

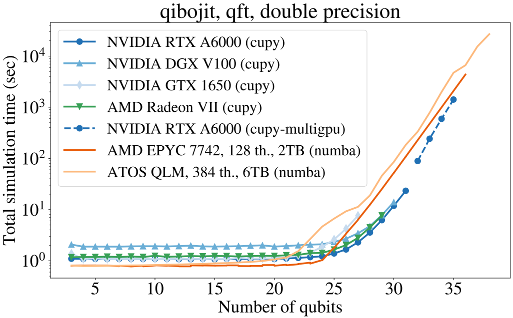
In-place updates.

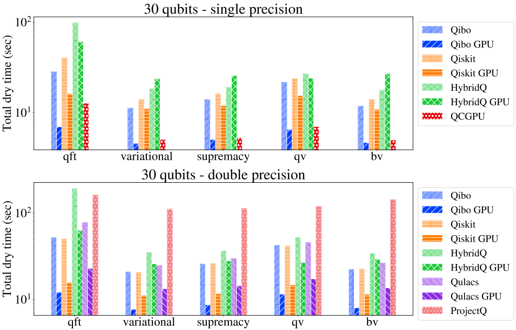
Specialized operators for 1 and 2 qubits gates exploiting sparsity.

A focus on classical simulation performances

arXiv:2203.08826

Through its modularity, `Qibo` allows execution of the same high level language onto different classical hardware accellerators.



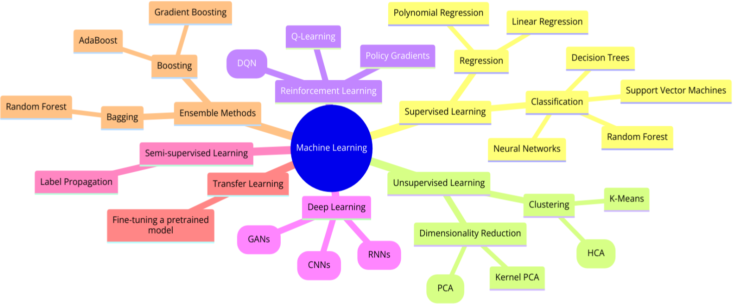


We reach satisfying performances thanks to custom operators and in-place updates of the statevector.

Quantum Machine Learning

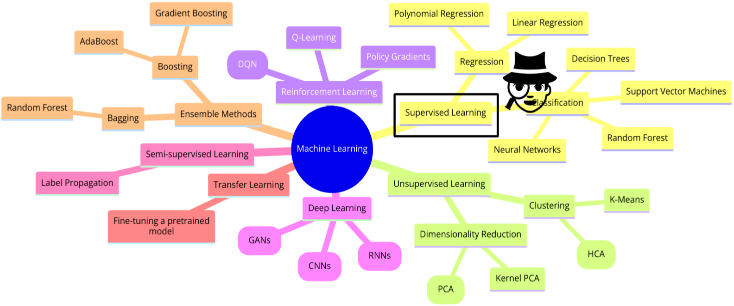
Classical Machine Learning

I asked ChatGPT to give me a comprehensive diagram of Machine Learning (ML) models.



Classical Machine Learning

I asked ChatGPT to give me a comprehensive diagram of Machine Learning (ML) models.

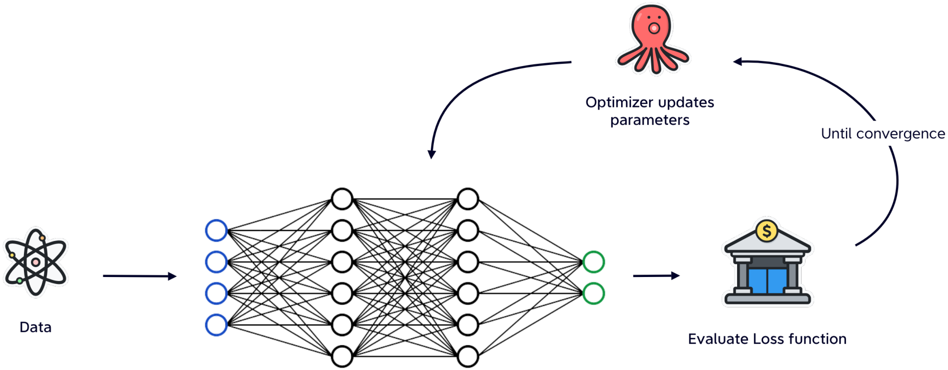


Gradient Boosting
AdaBoost
Boosting
Random Forest
Bagging
Ensemble Methods
Q-Learning
DQN
Policy Gradients
Reinforcement Learning
Polynomial Regression
Linear Regression
Regression
Decision Trees
Support Vector Machines
Classification
Supervised Learning
Random Forest
Neural Networks
Machine Learning
Semi-supervised Learning
Label Propagation
Transfer Learning
Fine-tuning a pretrained model
Deep Learning
GANs
CNNs
RNNs
Unsupervised Learning
Clustering
K-Means
HCA
Dimensionality Reduction
PCA
Kernel PCA

Focusing on the supervised ML!

- we aim to know some hidden law between two variables: $\boldsymbol{y} = f(\boldsymbol{x})$;
- we define a parameteric model which returns $\boldsymbol{y}_{\mathrm{est}} = f_{\mathrm{est}}(\boldsymbol{x};\boldsymbol{\theta})$;
- we define an optimizer, which task is to compute $\mathrm{argmin}_{\boldsymbol{\theta}}\left[J(\boldsymbol{y}_{\mathrm{meas}},\boldsymbol{y}_{\mathrm{est}})\right]$.

Classical Machine Learning



Optimizer updates parameters

Until convergence

Data

Evaluate Loss function

Quantum Machine Learning

Machine Learning

$\mathcal{M}$: model;

$\mathcal{O}$: optimizer;

$\mathcal{J}$: loss function.

(x, y): data

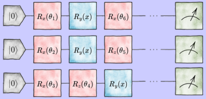
Quantum Computation

$\mathcal{Q}$: qubits;

$\mathcal{S}$: superposition;

$\mathcal{E}$: entanglement.

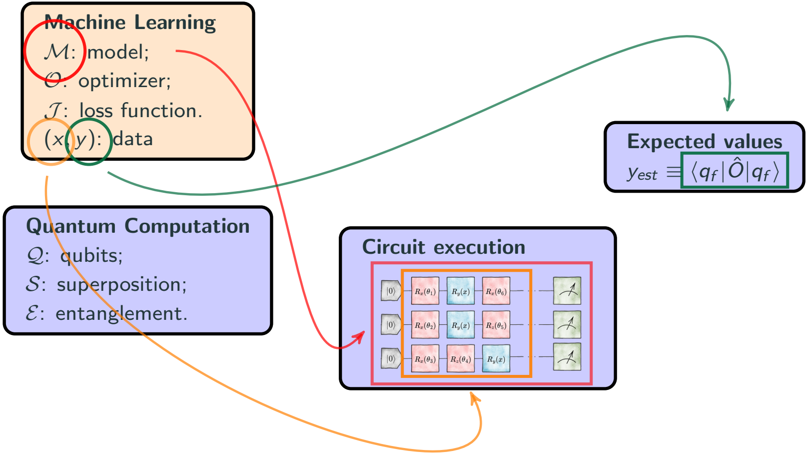
Circuit execution



Expected values

$y_{est} \equiv \langle q_f | \hat{O} | q_f \rangle$

Quantum Machine Learning



Machine Learning

$\mathcal{M}$: model;

$\mathcal{O}$: optimizer;

$\mathcal{J}$: loss function.

(x, y): data

Quantum Computation

$\mathcal{Q}$: qubits;

$\mathcal{S}$: superposition;

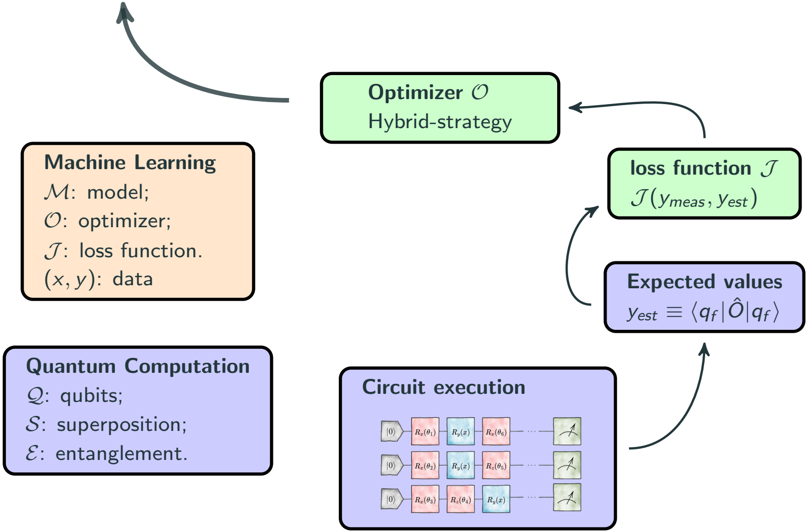
$\mathcal{E}$: entanglement.

Circuit execution

Expected values

$y_{est} \equiv \boxed{\langle q_f | \hat{O} | q_f \rangle}$

Quantum Machine Learning!



Optimizer $\mathcal{O}$
Hybrid-strategy

Machine Learning
$\mathcal{M}$: model;
$\mathcal{O}$: optimizer;
$\mathcal{J}$: loss function.
(x, y): data

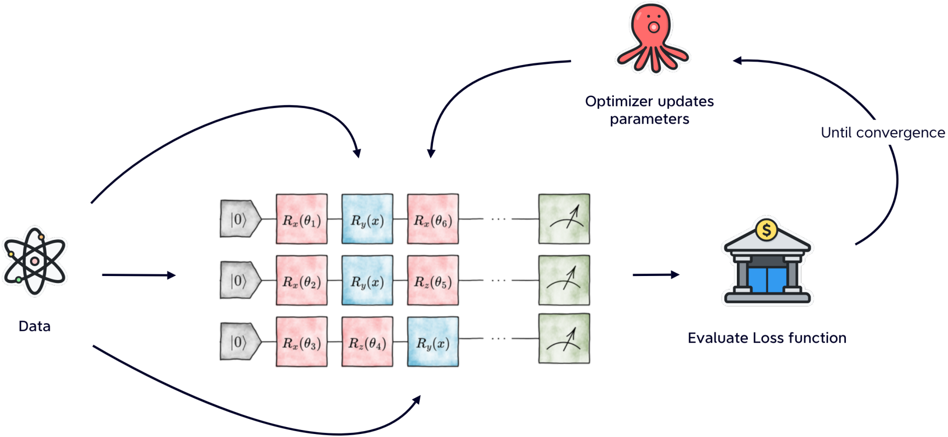
loss function $\mathcal{J}$
$\mathcal{J}(y_{meas}, y_{est})$

Expected values
$y_{est} \equiv \langle q_f | \hat{O} | q_f \rangle$

Quantum Computation
$\mathcal{Q}$: qubits;
$\mathcal{S}$: superposition;
$\mathcal{E}$: entanglement.

Circuit execution

From ML to QML



Optimizer updates parameters

Until convergence

Data

Evaluate Loss function

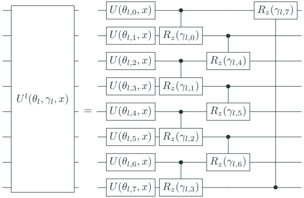
>_ Example 2: PDF fit

arXiv:2011.13934

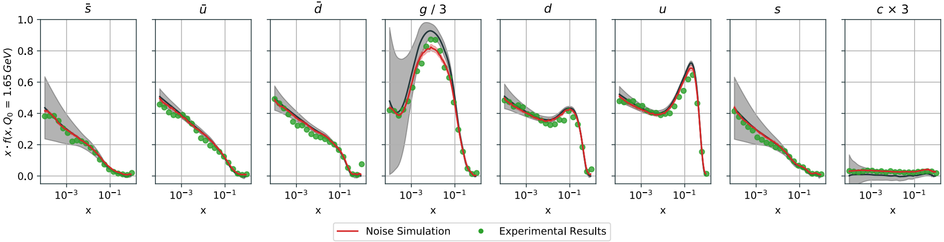
We parametrize **Parton Distribution Functions** with multi-qubit variational quantum circuits:

1. Define a quantum circuit:
   $\mathcal{U}(\theta, x)|0\rangle^{\otimes n} = |\psi(\theta, x)\rangle$
2. $U_w(\alpha, x) = R_z(\alpha_3 \log(x) + \alpha_4) R_y(\alpha_1 \log(x) + \alpha_2)$
3. Using $z_i(\theta, x) = \langle\psi(\theta, x)|Z_i|\psi(\theta, x)\rangle$:

$$\mathrm{qPDF}_i(x, Q_0, \theta) = \frac{1 - z_i(\theta, x)}{1 + z_i(\theta, x)}.$$



Results from **classical quantum simulation and hardware execution** (IBM) are **promising**:



PDF fit on chip

arXiv:2308.06313

High level API: Qibo

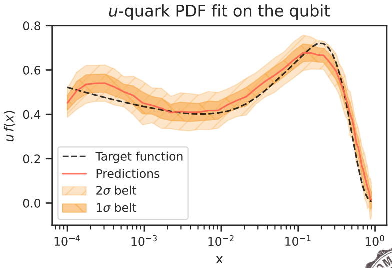
- define **prototypes** and models;
- **simulate** training and noise.

Calibration: Qibocal

- **calibrate** qubits;
- generate **platform configuration**;

Execution: Qibolab

- allocate **calibrated** platform;
- **compile** and **transpile** circuits;
- execute and return **results**.



HOME MADE

Parameter	Value
N_{data}	50
N_{shots}	500
MSE	$\sim 10^{-3}$
Electronics	Xilinx ZCU216
Training time	$\sim$ 2h

Some applications

- Multi-variable integration using the qPDF ansatz, 🕮 arXiv:2303.11346;
- Real-time quantum error mitigation on superconducting devices to improve trainability, 🕮 arXiv:2303.11346;