Towards a hybrid quantum operating system

Andrea Pasquale on the behalf of the Qibo collaboration

30th October 2023, ML_INFN

Technology Innovation Institute

Table of contents

What is Quantum Computing?

“Nature isn’t classical dammit, and if you want to make a simulation of Nature you better make it quantum mechanical, and by golly it’s a wonderful problem because it doesn’t look so easy.”

Richard Feynman

Quantum Superposition

Compared to classical bits which can be represented by a single state (0 or 1), quantum bits can be prepared in any superposition of $|0\rangle$ or $|1\rangle$.

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

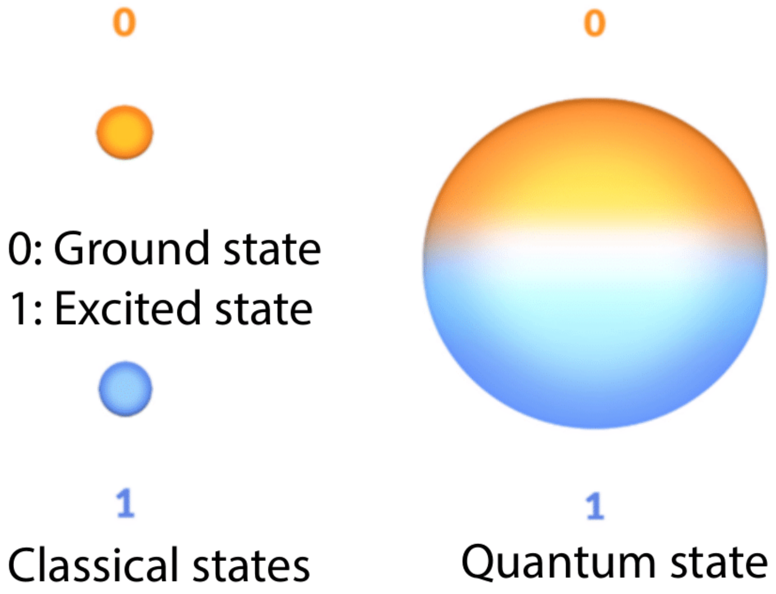
where $\alpha, \beta \in \mathcal{C} : |\alpha|^2 + |\beta|^2 = 1$.

In the case of a system with two qubit we obtain the following representation:

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$$

with a similar normalization condition:

$$\sum_{i,j=0,1} |\alpha_{ij}|^2 = 1$$



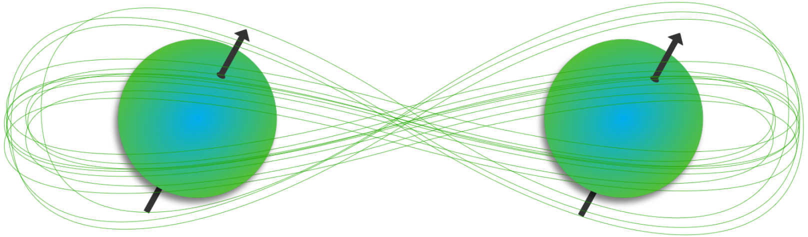
Quantum Entanglement

In QC we can extract correlations between different qubits through *entanglement*.

For example if we consider the following state[1]:

$$|\Phi^+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

whenever we measure $|0\rangle$ for the first qubit also the second one will be measured in $|0\rangle$ and the same goes for $|1\rangle$.



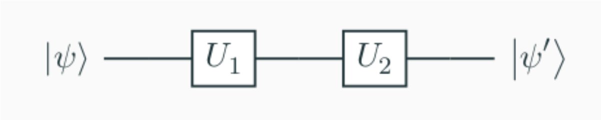
1. This state is known as Bell state, an example of maximally entangled state.

How to modify the state of a Qubit?

Being a quantum system qubits $|\psi\rangle$ evolve over time through unitary operators U_i.

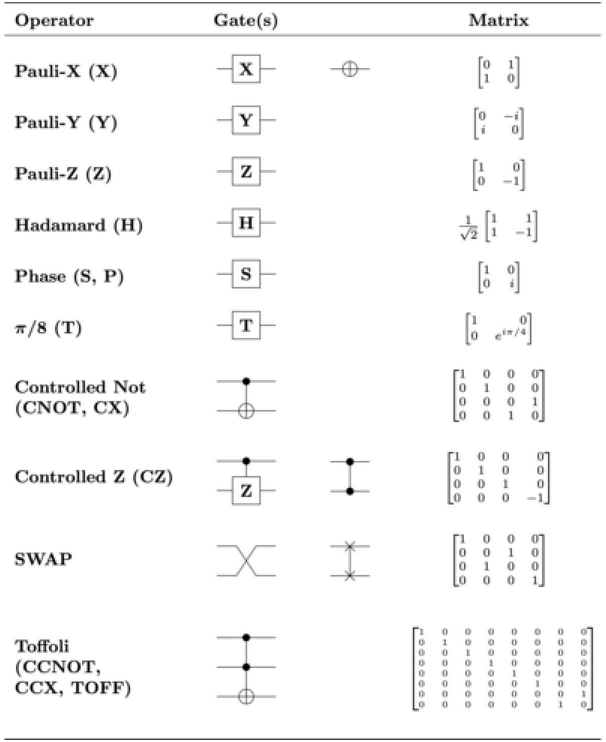
$$|\psi'\rangle = U_1 U_2 |\psi\rangle$$

Maintaining an analogue to classical quantum we can represent the following evolution as a small quantum circuit:



which we call quantum circuit.

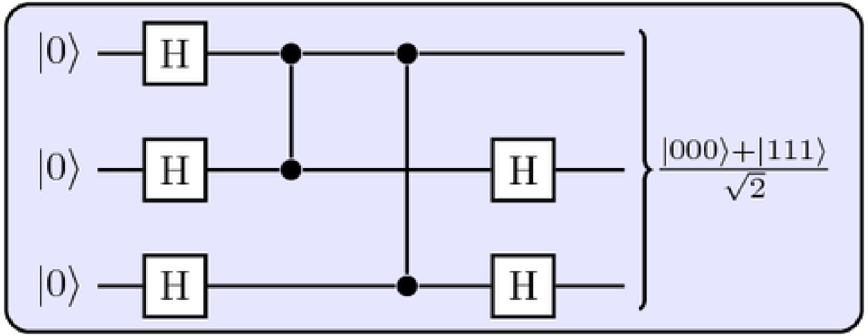
Contrary to classical circuit only reversible gates can be used in quantum circuits



Operator	Gate(s)	Matrix
Pauli-X (X)	X	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y (Y)	Y	$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli-Z (Z)	Z	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard (H)	H	$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
Phase (S, P)	S	$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$
$\pi/8$ (T)	T	$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$
Controlled Not (CNOT, CX)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
Controlled Z (CZ)	Z	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$
SWAP		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Toffoli (CCNOT, CCX, TOFF)		$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$

Quantum circuits

A quantum circuits with more than one qubit can be represented as:



Exponential scaling

All the possible initial states for a system of 3 qubits are 2^3, in fact a generic unitary for this system is 8×8 matrix.

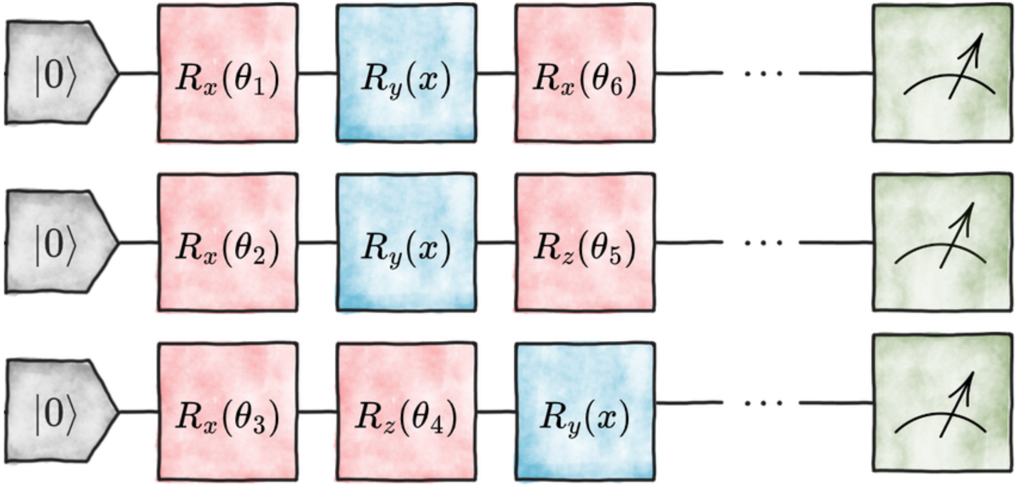
Increasing the number of qubits leads to exponential scaling of the system $\Rightarrow$ more expressivity!

A snapshot of Quantum Machine Learning

Variational Quantum Circuits

How can we encode information in a quantum circuits?

Variational Quantum Circuit: quantum circuits with parametrized gates.

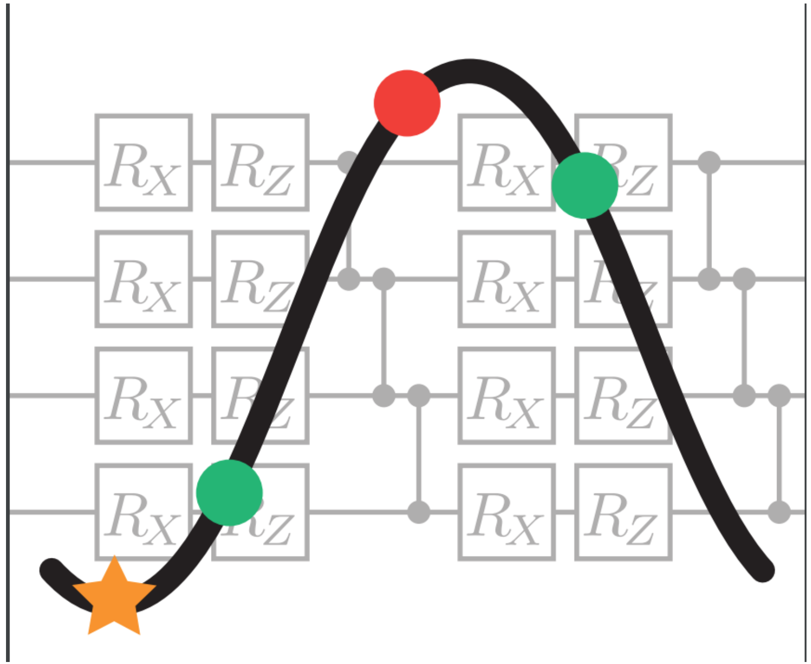


Observation: We can use the Variational Quantum Circuits as as a neural network.

Rational

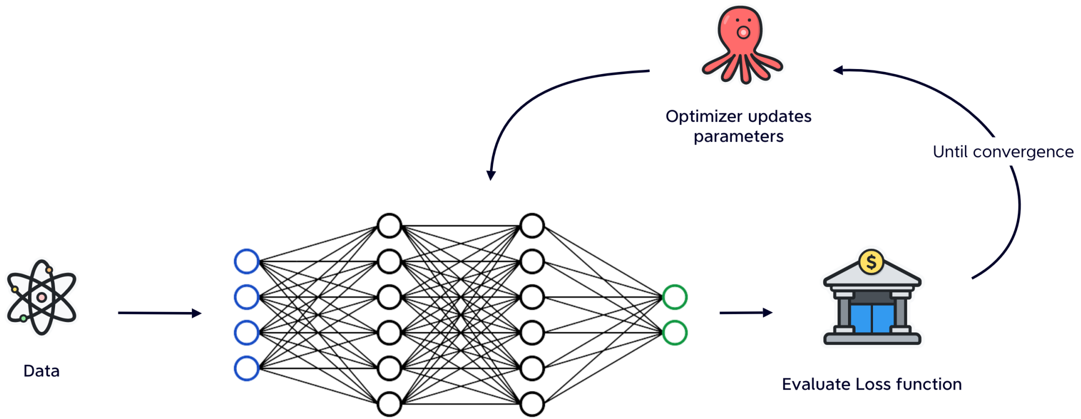
Using Variational Quantum Circuits we can define a Variational Quantum Computer!

1. we want a quantum circuit $\mathcal{U}(\theta)$ to approximates some law V.
2. executing $\mathcal{U}(\theta)$ we use a variational quantum state to reach the solution
3. Solovay-Kitaev theorem: the number of gates needed by $\mathcal{U}$ to represent V with precision δ is $\mathcal{O}(\log^c \delta^{-1})$, where $c < 4$.



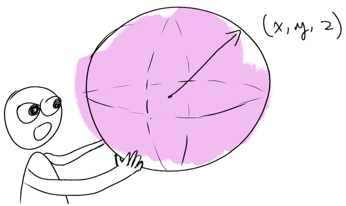
ML workflow for QML

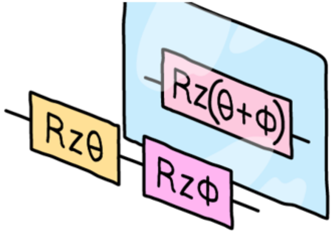
Now we can treat a VQC as a neural network in order to optimize the parameters through back-propagation.

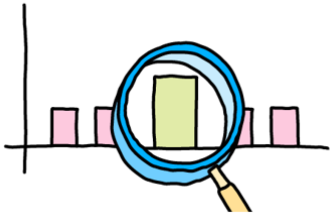


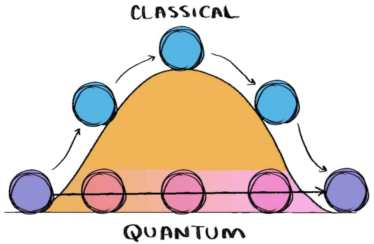
Why QML?

- shallow models thanks to superposition and entanglement
- map problems into Hilbert's spaces loads to high expressivity
- exploit QC sub-routines to speed-up classical algorithms (e.g. using Grover)
- physical advantages when dealing with combinatorial optimization (quantum annealing)





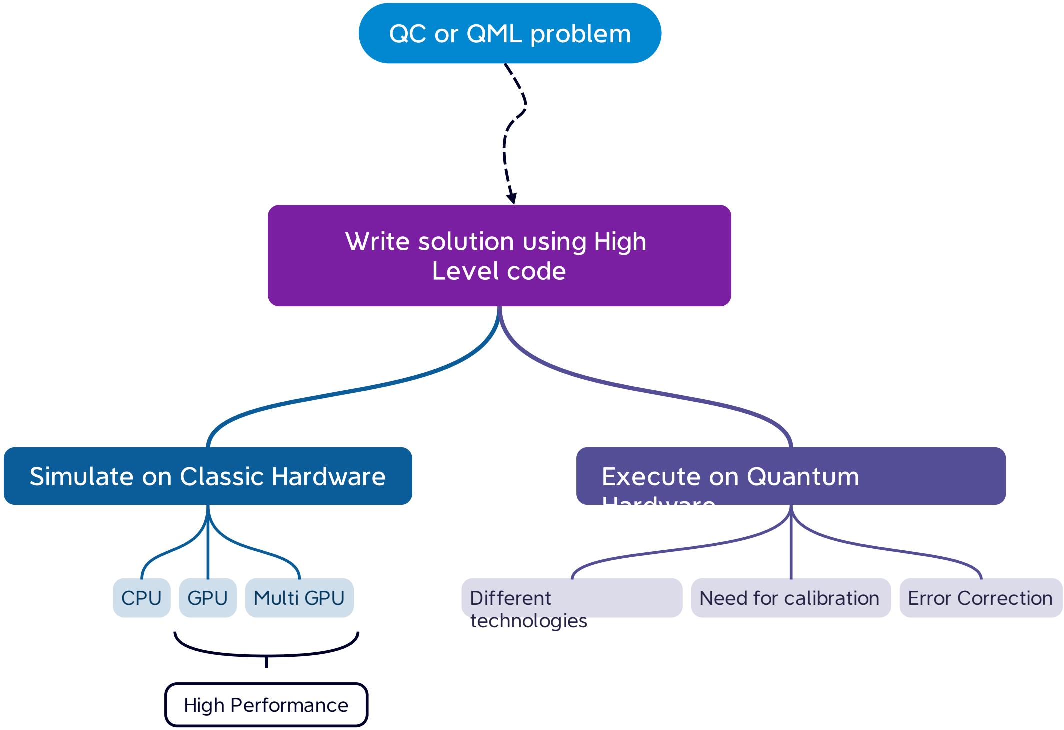




Introducing Qibo

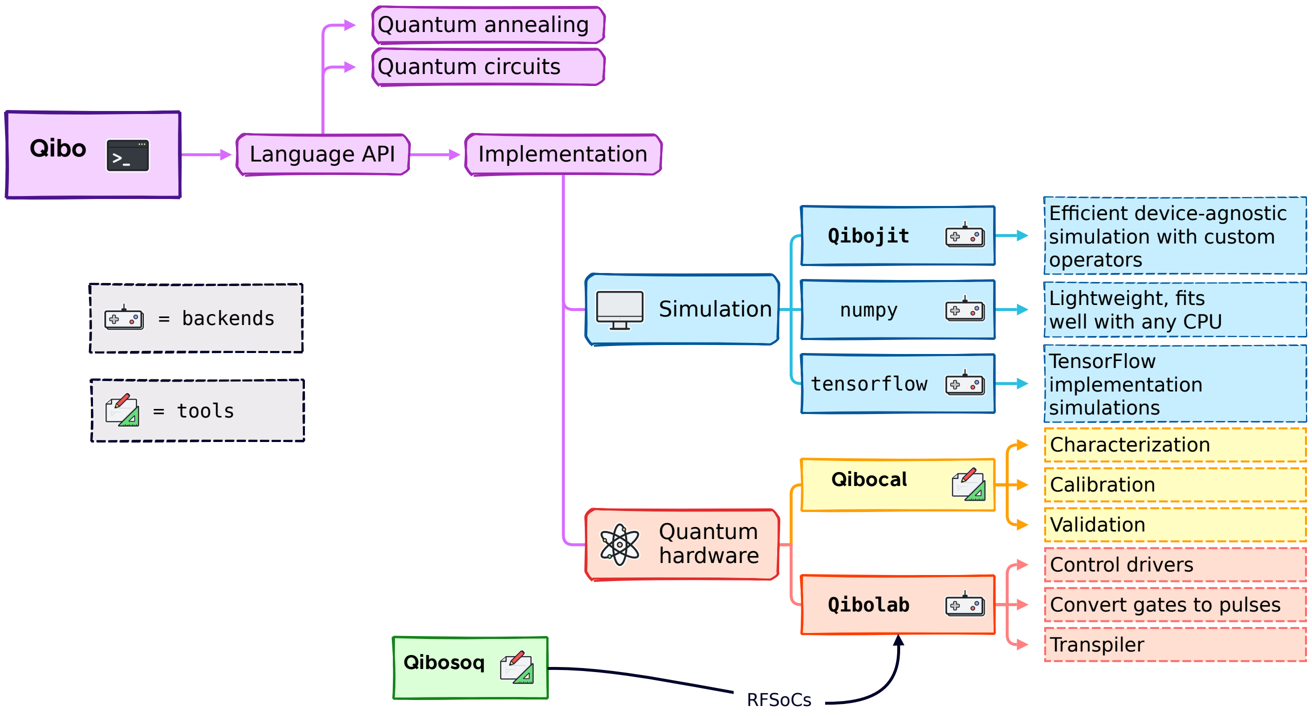
Open-source full stack API for quantum simulation, hardware control and calibration

Why do you need a framework?



Is it possible to create from scratch a framework for all of this?

QIBO: A BRIEF OVERVIEW



Gate set abstraction

```python
import numpy as np
from qibo.models import Circuit
from qibo import gates, set_backend

# Set driver engine
set_backend("numpy")

c = Circuit(2)
c.add(gates.X(0))

# Add a measurement register on both qubits
c.add(gates.M(0, 1))

# Execute the circuit with the default initial state |00>.
result = c(nshots=100)
```

```python
# Change backend
set_backend("qibojit")

# Circuit execution with new driver
result = c(nshots=100)
```

Qibo features

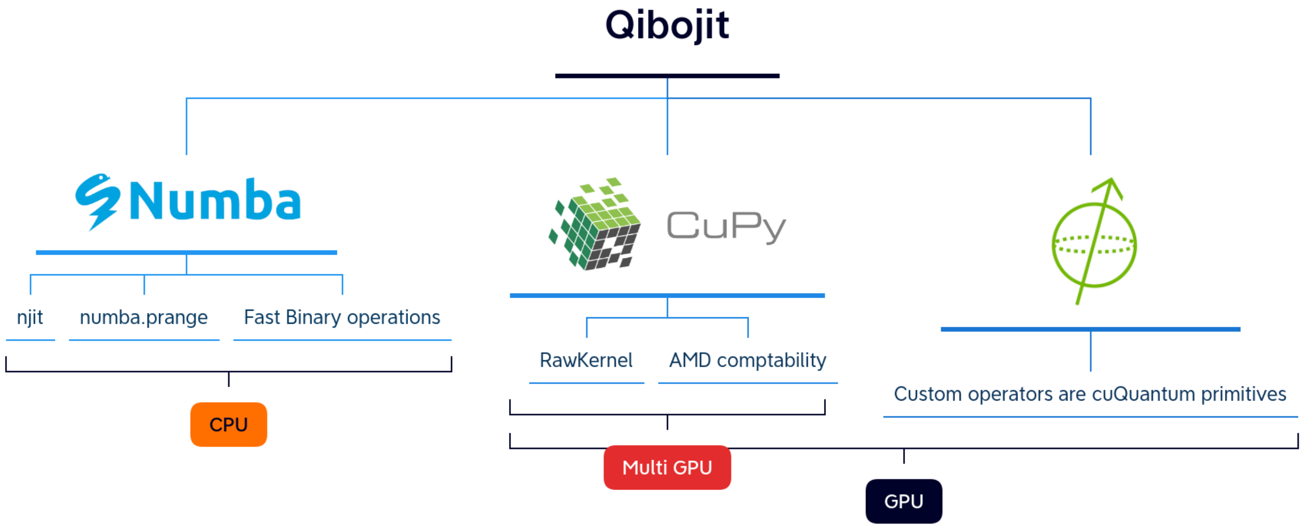
- Definition of a standard language for the construction and execution of quantum circuits with device agnostic approach to simulation and quantum hardware control based on plug and play backend drivers.
- A continuously growing code-base of quantum algorithms and applications presented with examples and tutorials.
- Efficient simulation backends with GPU, multi-GPU and CPU with multi-threading support.
- A simple mechanism for adding new simulation and hardware backend drivers.

2009.01845

High performance simulation

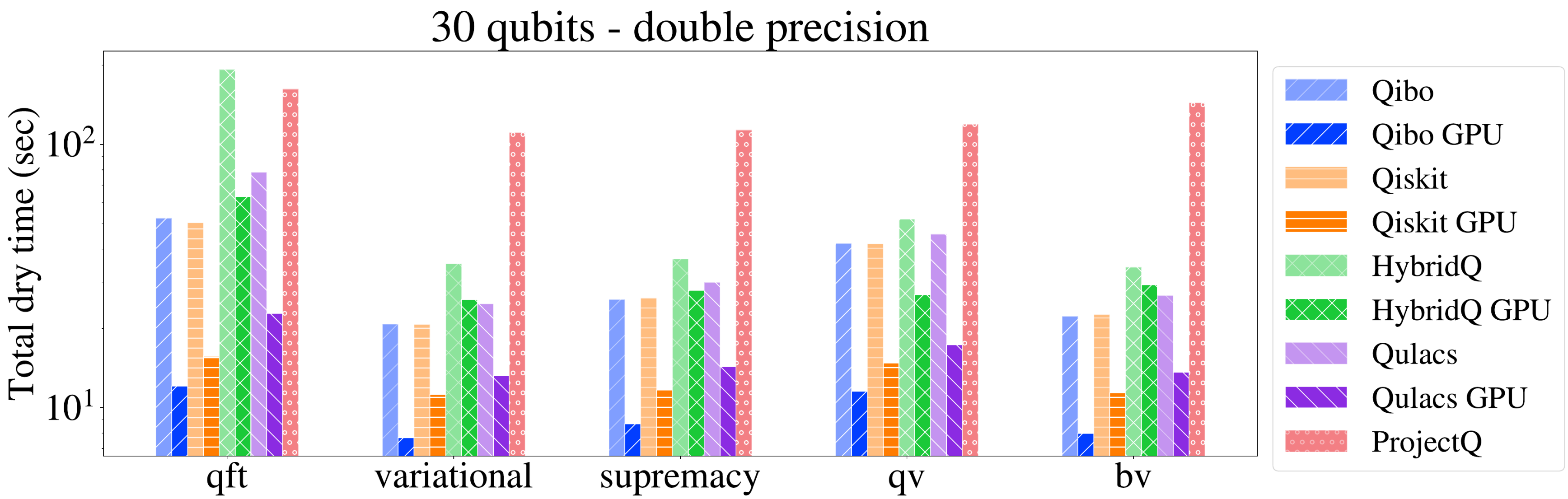
❌ Long computational times using naive approach (`Numpy` or `TensorFlow`) for circuits with large number of qubits.

✅ We need more sophisticated tools to be able to simulate a quantum circuits with more qubits!



2203.08826

Benchmark

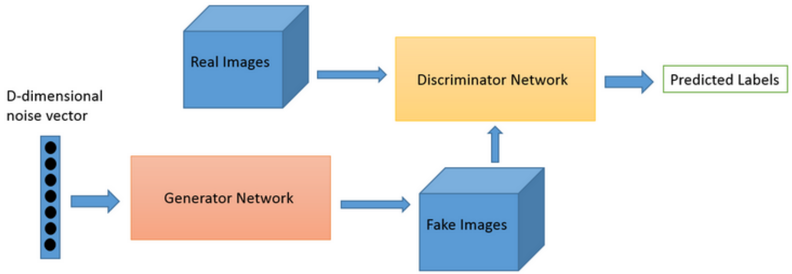


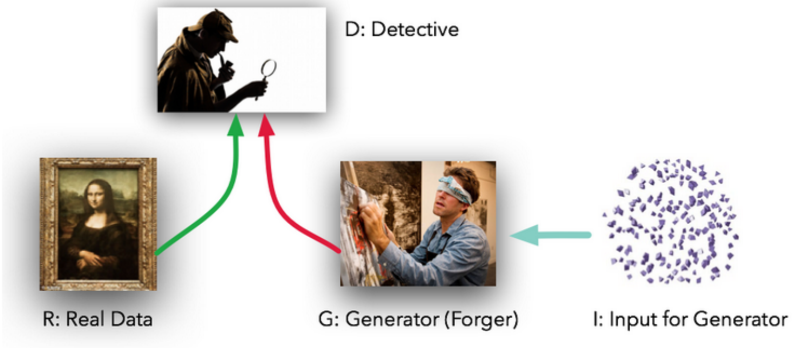
All the benchmarks are available in qibojit-benchmarks.

Quantum Machine Learning examples using Qibo

Monte Carlo event generator using Quantum GAN

What are Generative Adversarial Networks?





Training

Adapt alternatively the generator $G(\phi_g, z)$ and the discriminator $D(\phi_d, x)$

Metrics

Binary cross-entropy for the loss functions:

$$\mathcal{L}_G(\phi_g, \phi_d) = -\mathbb{E}_{z \sim p_{\text{prior}}(z)}[\log D(\phi_d, G(\phi_g, z))]$$

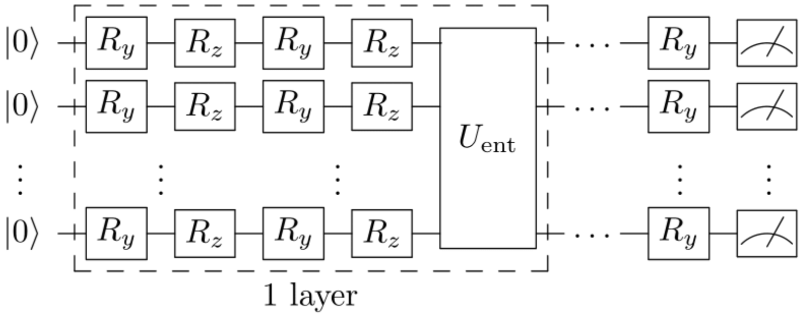
$$\mathcal{L}_D(\phi_g, \phi_d) = \mathbb{E}_{x \sim p_{\text{real}}(x)}[\log D(\phi_d, x)] + \mathbb{E}_{z \sim p_{\text{prior}}(z)}[\log(1 - D(\phi_d, G(\phi_g, z)))]$$

Game theory: min-max two-player game to reach Nash equilibrium

$$\min_{\phi_g} \mathcal{L}_G(\phi_g, \phi_d) \quad \max_{\phi_d} \mathcal{L}_D(\phi_g, \phi_d)$$

A CLASSICAL-QUANTUM APPROACH

We replace the classical generator using a VQC:



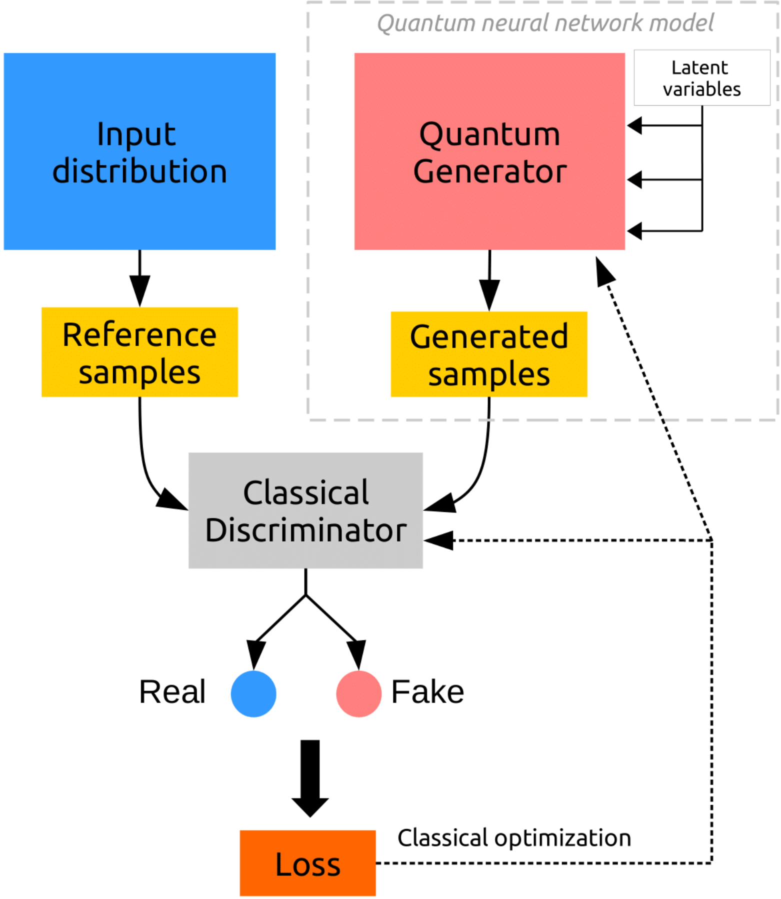
with the following encoding of the noise:

$$R_{y,z}^{l,m}\left(\vec{\phi}_g, \vec{z}\right) = R_{y,z}\left(\phi_g^{(l)} z^{(m)} + \phi_g^{(l-1)}\right)$$

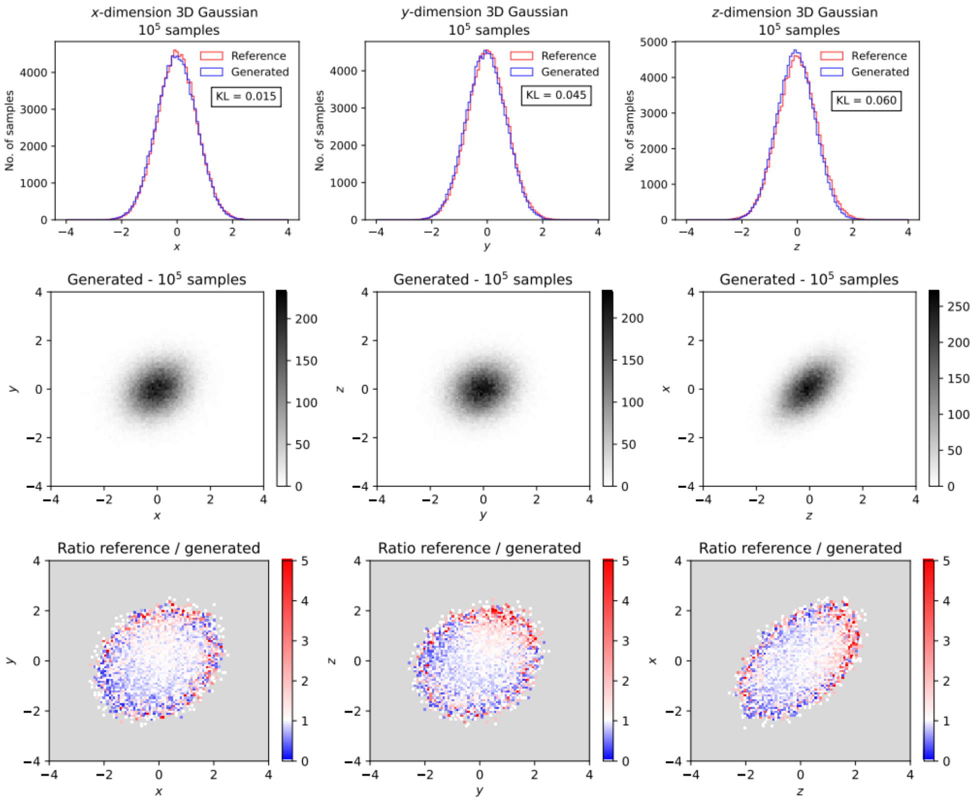
we sample according to

$$x_i = \langle \Psi_{\phi_g}(\vec{z}) | \, \sigma_z^i \, | \Psi_{\phi_g}(\vec{z}) \rangle$$

2110.06933

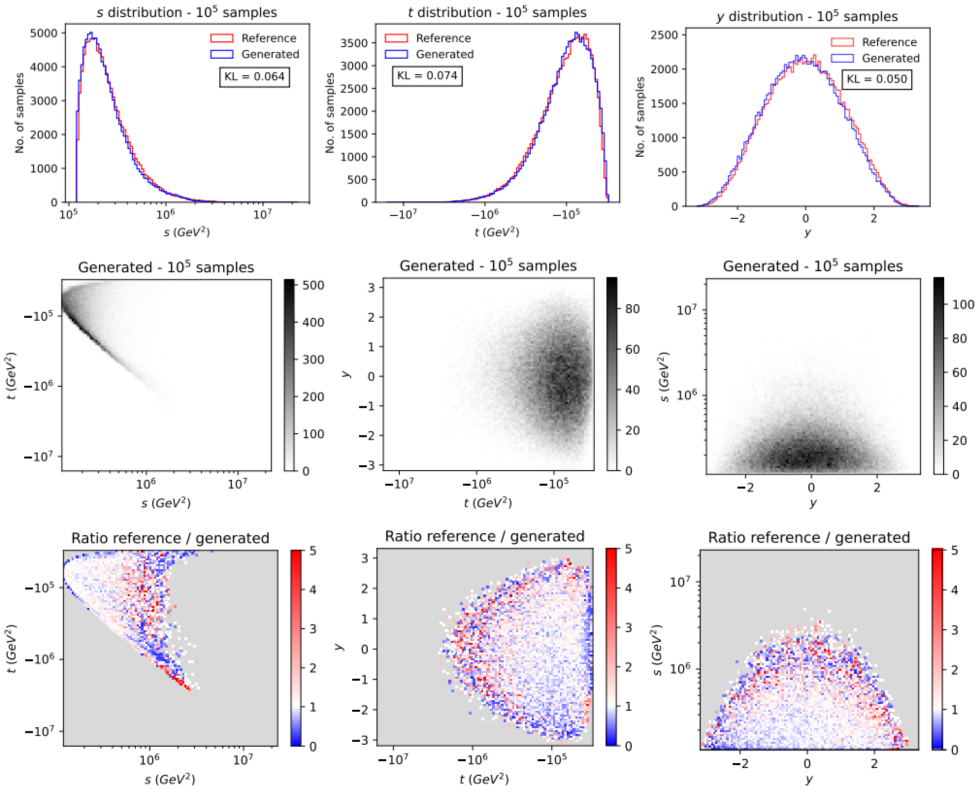


Validation: 3D correlated Gaussian function



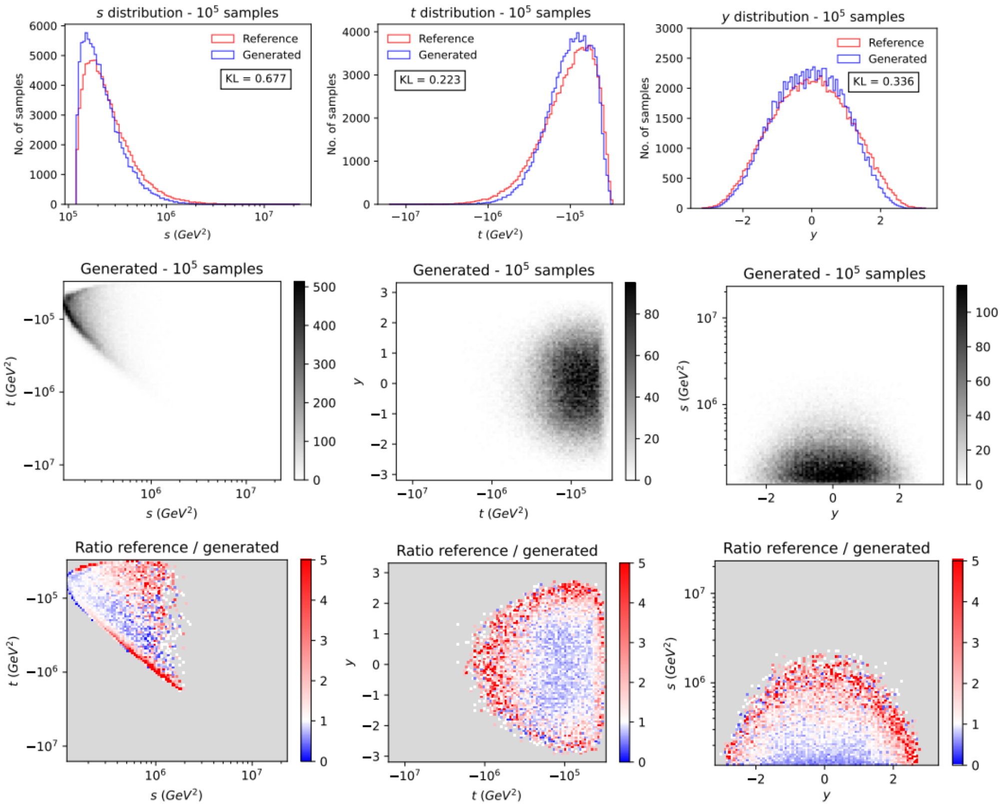
A MORE CHALLENGING EXAMPLE: $pp \to t\bar{t}$

Simulation



RESULTS WITH $pp \rightarrow t\bar{t}$ EXECUTION ON HARDWARE

Execution on Hardware

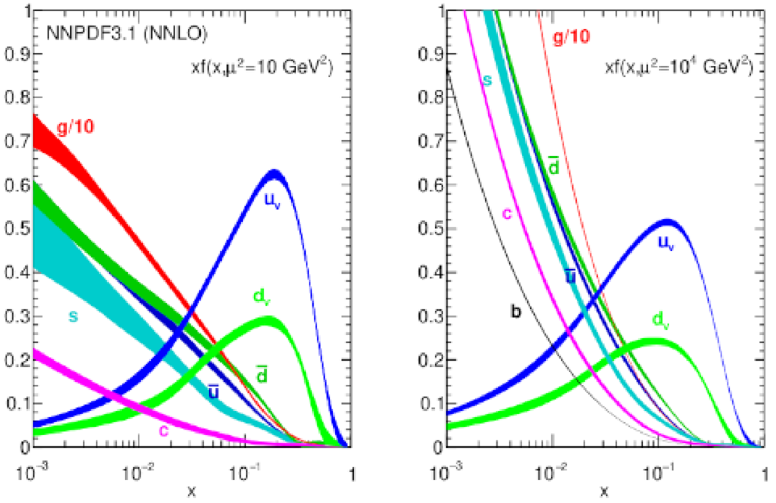


Determining the proton content with a quantum computer

In this work a parametrized Variational Quantum Circuit (VQC) is employed to fit Parton Density Functions (PDFs)[1]:

$$\mathrm{qPDF}_i(x, Q_0, \theta)$$

where x is the momentum fraction of the hadron carried by the parton i at fixed energy scale Q_0 while θ are parameters of the VQC.



2011.13934

This is just one of the possible application of QC/QML in High Energy Physics!

Methodology

The model is created as follows:

1. Inject x in VQC: $\mathcal{U}(\theta) \rightarrow \mathcal{U}(\theta, x)$[1]
2. Extract information from QC through series of Hamiltonians:

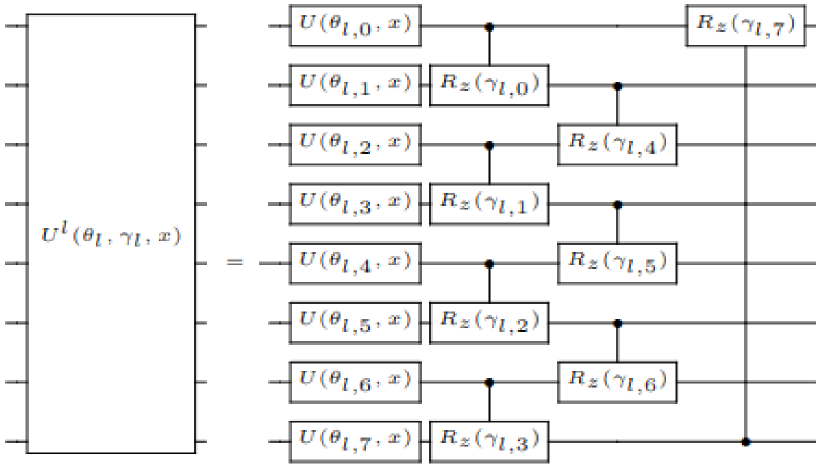
$$Z_i = \bigotimes_{j=0}^{n} Z^{\delta_{ij}}$$

3. Define the function:

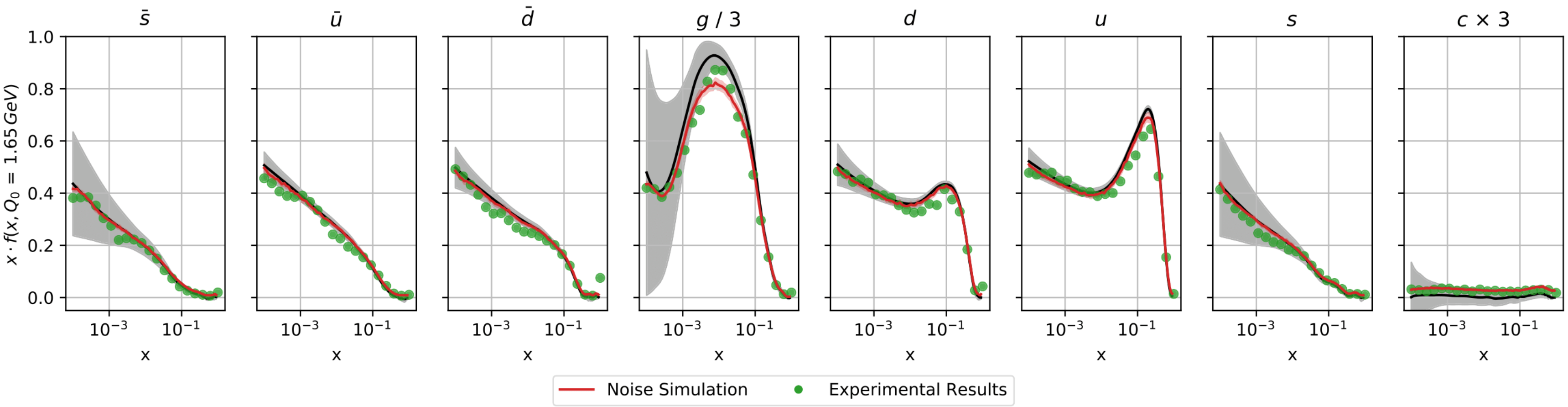
$$z_i(\theta, x) = \langle \psi | \mathcal{U}^\dagger(\theta, x) Z_i \mathcal{U}(\theta, x) | \psi \rangle$$

4. Finally we define the qPDF model for flavour i at a given (x, Q_0) as

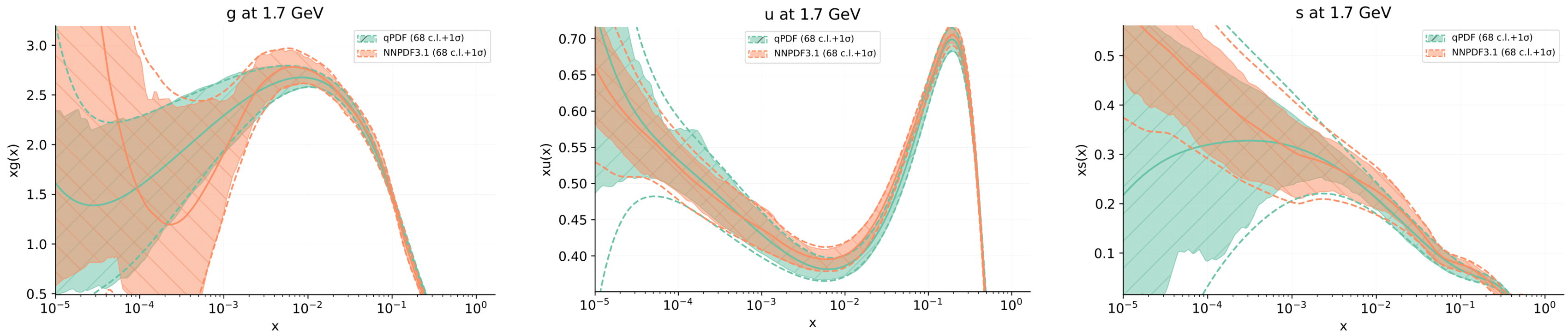
$$\text{qPDF}_i(x, Q_0, \theta) = \frac{1 - z_i(\theta, x)}{1 + z_i(\theta, x)}$$



Representation of a single layer used.



Top: Single-flavour fit for all flavours for 5 layers and 8 qubits. The red lines are the prediction of the qPDF model with simulated noise from IBM processor. Green points are results from running on actual quantum hardware from IBM. Bottom: Fit results for the gluon and the u and s quarks. qPDF is able to reproduce the features of NNPDF3.1. We now see this is also true when the fit performed by comparing to data and not by comparing directly to the goal function. The differences seen at low-x can be attributed to the lack of data in that region.



Probability density function determination using adiabatic quantum annealing

2303.11346

The goal of this work is to estimate the probabilitiy density function value $\rho(x)$ for each element of a sample of data $\omega = \{x_i\}_{i=1}^{N_{data}}$ using the following Quantum Adiabatic Machine Learning (QAML) strategy:

- encoding the CDF values $F(x)$ inoto an adiabatic evolution
- translating the adiabatic Hamiltonian into a circuit $\mathcal{C}$ callable at any time τ
- Derivating the circuit using the parameter shift rule[1] obtaining the PDF

1. The Parameter Shift Rule (PSR)[https://arxiv.org/abs/1905.13311] is an algorithm to compute the derivative of specific quantum circuits which is hardware compatible

Model regression with QAML

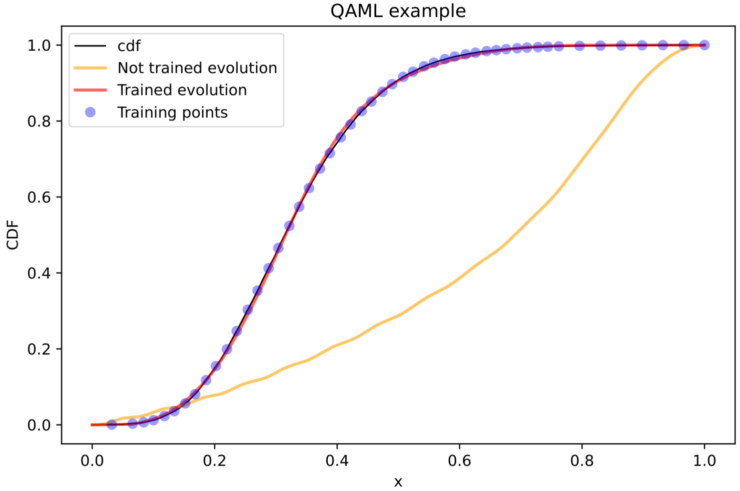
Given a one-dimensional function $f(t)$ we can choose two Hamiltonians H_0 and H_1 such that we have $f(t_0) = \langle H_0 \rangle$ and $f(t_1) = \langle H_1 \rangle$. Therefore we need to find a time-dependent Hamiltonian $H(t)$ such that

$$\langle H(t) \rangle = f(t)$$

We can create a parametric model for this Hamiltonian by using quantum annealing with a parametric scheduler:

$$H(t) = \Big[1 - s(t,\theta)\Big] H_0 + s(t,\theta) H_1$$

We can then use standard ML tools to train $H(t)$ by optimizing the parameters θ in the scheduling $s(t,\theta)$.



Example of initial and final state of the algorithm. The Ntrain blue points are the training set selected from a gaussian mixture sample, whose empirical CDF is represented by the black line. The random initialization of the adiabatic evolution leads to the initial sequence of energies (yellow line). After a training time, the evolution is closer to the training set (red line).

Translation of Hamiltonian into derivable circuit

Sketch of the algorithm

1. Diagonalization of each H_j

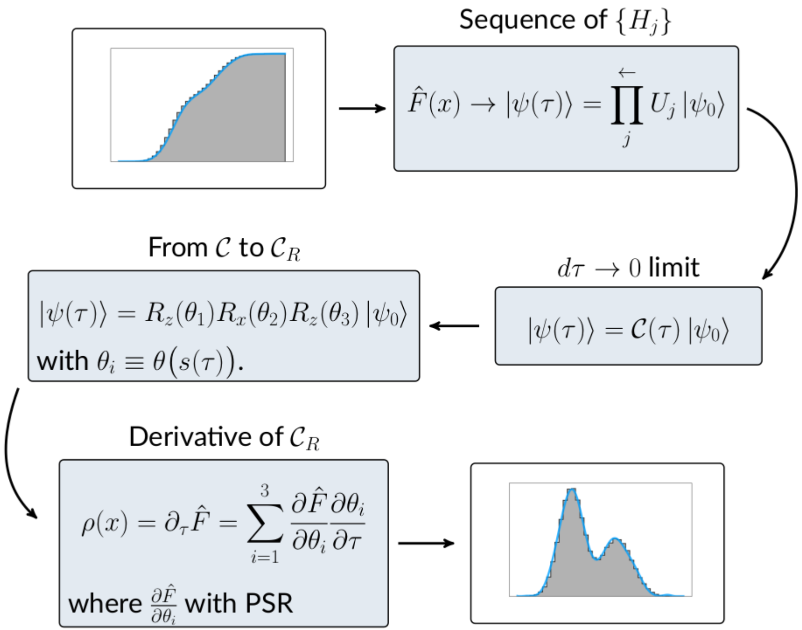
$$\mathcal{C}_n = \prod_{j=0}^{n} e^{id\tau H_j} = \prod_{j=0}^{n} P_j e^{id\tau D_j} P_j^{-1}$$

2. Take the limit $d\tau \to 0$

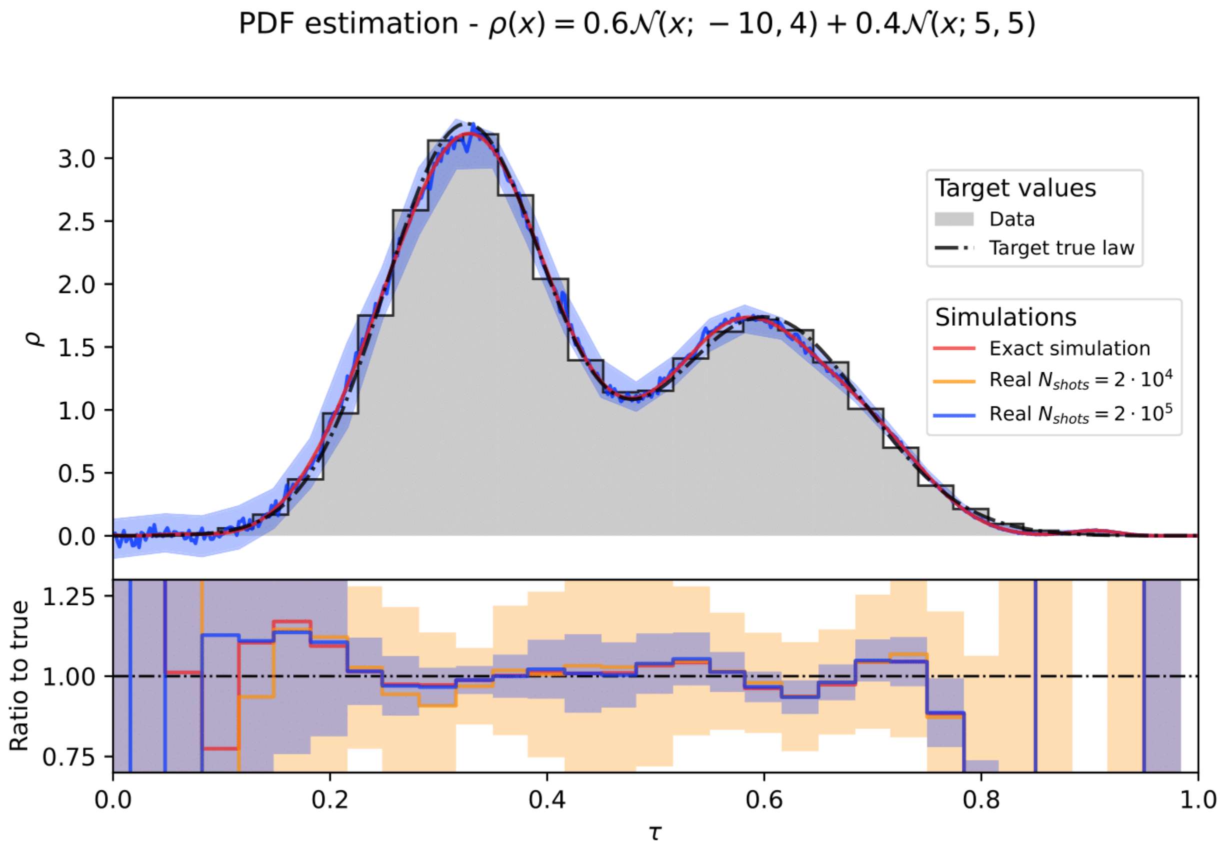
$$C(t) = P_t \exp\left(i \int_0^{t/T} D_j d\tau\right) P_0^{-1}$$

3. Rewrite circuit as compositon of rotations and apply PSR

$$\mathrm{PDF}(t) = \mathrm{PSR}\langle \mathcal{C}(t)^\dagger Z \mathcal{C}(t) \rangle$$



Validation


PDF estimation - $\rho(x) = 0.6\mathcal{N}(x; -10, 4) + 0.4\mathcal{N}(x; 5, 5)$
Target values
Data
Target true law
Simulations
Exact simulation
Real $N_{shots} = 2 \cdot 10^4$
Real $N_{shots} = 2 \cdot 10^5$
ρ
Ratio to true
τ
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0.75
1.00
1.25
0.2
0.4
0.6
0.8

Introducing Qibolab

Quantum control for self-hosted QPUs using Qibo

Motivation I: Gate to Pulse conversion

Usually a quatum computer will be able to produce only a few gates called native gates through a specific series of pulses generated by dedicated instruments. A physical implementation of a quantum gate requires the conversion of a matrix to a sequence of microwave signals:

$$\sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Conversion to Pulse API

```
RX:
    duration: 30
    amplitude: 0.5
    frequency: 4_939_827_950
    shape: Rectangular()
    type: qd
    start: 0
    phase: 0
```

Instrument specific implementation

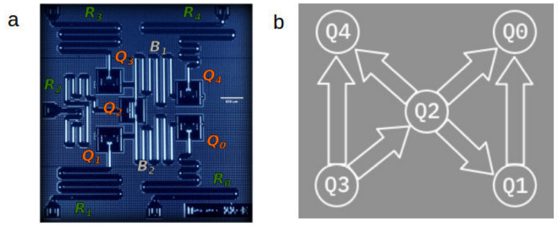
It can be shown that any arbitrary single qubit can be written in terms on native gates R_X and R_Z:

$$U_3(\theta, \phi, \lambda) = \begin{pmatrix} \cos\frac{\theta}{2} & -e^{i\lambda}\sin\frac{\theta}{2} \\ e^{i\phi}\sin\frac{\theta}{2} & e^{i(\phi+\lambda)}\cos\frac{\theta}{2} \end{pmatrix} \rightarrow R_Z(\phi)R_X(-\pi/2)R_Z(\phi)R_X(\pi/2)R_Z(\phi)$$

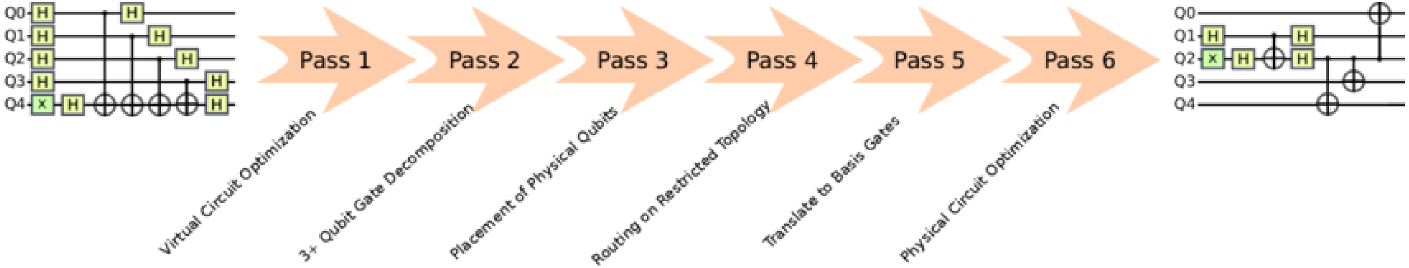
This proof can be generalized for multi-qubit gates.

Motivation II: Circuit transpilation

Usually a quantum computer will have a specific connectivity:



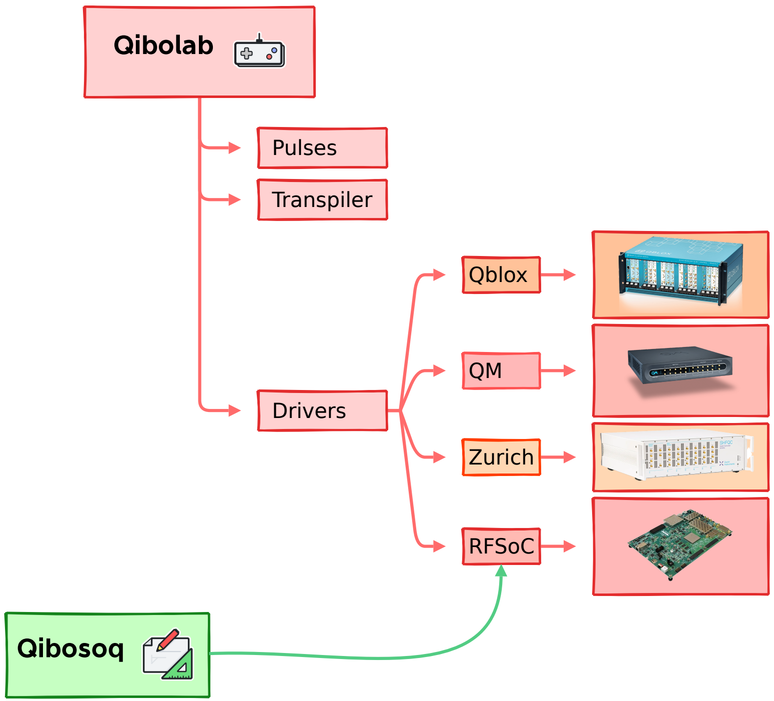
To be able to execute arbitrary circuits we need to rewrite the circuit in a way that matches the topology of the specific quantum devices. Therefore we need also all the tools to be able to perform these graph manipulations which usually we refer to as transpiler.



QIBOLAB API

Some of the key features of Qibolab are:

- Platform API: support custom allocation of quantum hardware platforms/ lab setup.
- Drivers: support commercial and open-source firmware for hardware control
- Pulse API: provide a library of custom pulses for execution through instruments
- Quantum circuit deployment: seamlessly deploys quantum circuit models on quantum hardware



2308.06313

2310.05851

Pulse API example

```python
from qibolab import create_platform
from qibolab.pulses import ReadoutPulse, PulseSequence

# Define PulseSequence
sequence = PulseSequence()
# Add some pulses to the pulse sequence
sequence.add(ReadoutPulse(start=0,
                          amplitude=0.3,
                          duration=4000,
                          frequency=200_000_000,
                          shape='Gaussian(5)'))

# Define platform
platform = create_platform("tii1q_b1")
# Platform setup
platform.connect()
platform.setup()
platform.start()
# Executes a pulse sequence.
results = platform.execute_pulse_sequence()
platform.stop()
platform.disconnect()
```

Drivers implemented

Currently Qibolab supports the following drivers:

- Qblox
- Quantum Machines
- Zurich Instruments
- RFSoC (based on Qick)

We also support local oscillators

- RohdeSchwarz SGS100A
- ERASynth

Qibolab

Introducing Qibocal

Quantum calibration and characterization using Qibo

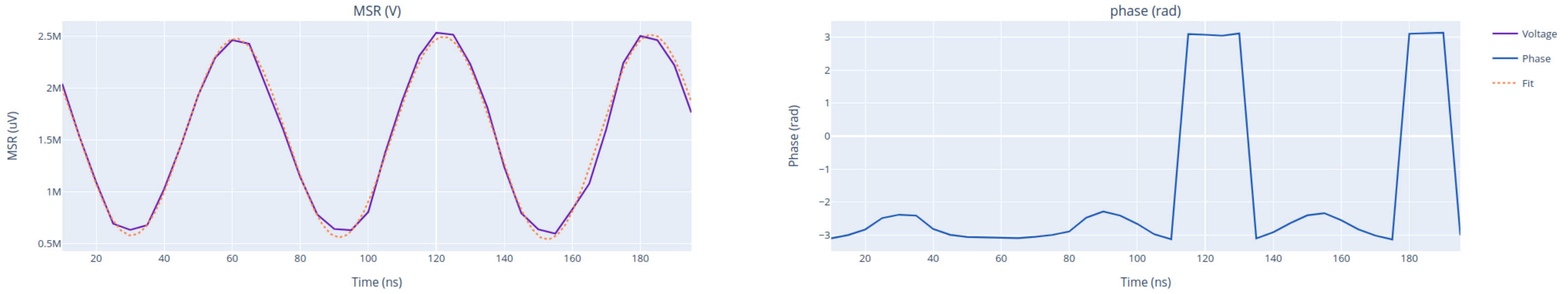
Motivation

Let's suppose the following:

1. We have a QPU (self-hosted).
2. We have control over what we send to the QPU.
3. We know how to convert quantum circuits to pulses.

Can I **trust** my results? **NO!**

Characterization and **calibration** are an essential step to properly operate emerging quantum devices.

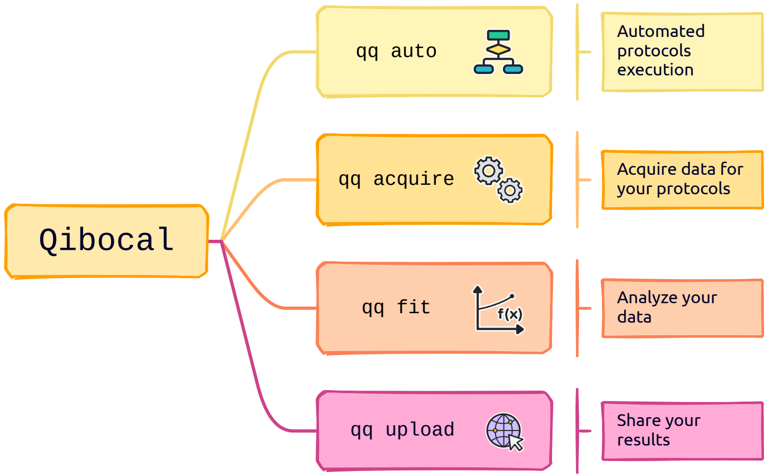


Calibration of RX pulse amplitude through a Rabi experiment through Qibocal.

QIBOCAL API

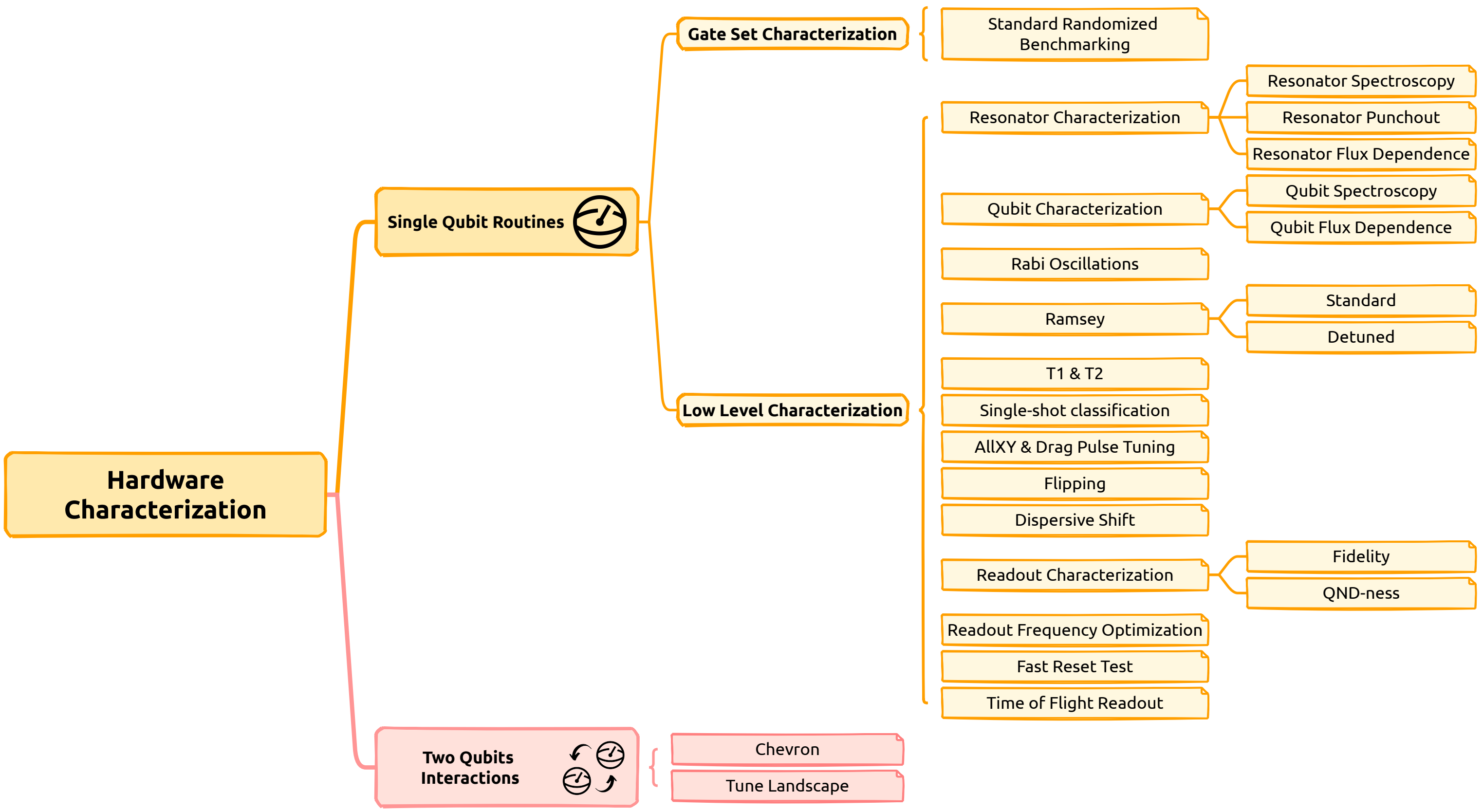
Qibocal key features:

- Automation of calibration protocol
- Declarative inputs using runcards
- Generation of HTML reports
- API to run protocols directly in python



2303.10397

Qubit characterization



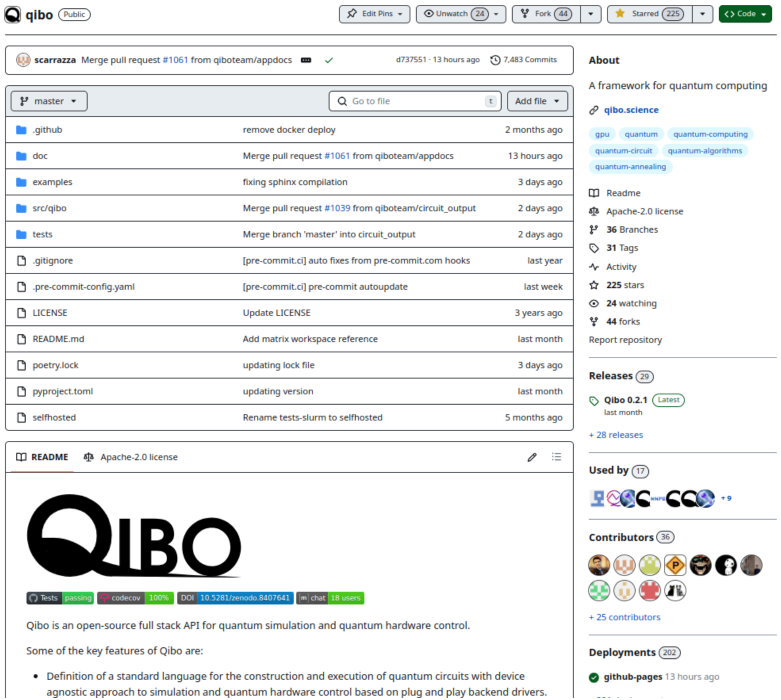
Outlook

We have presented Qibo, a simple full stack API capable of

- Deploying QML algorithms:
- Simulating large circuits in an efficient way: qibojit
- Deploying circuits in self-hosted QPUs: qibolab
- Calibrating self-hosted QPUs: qibocal

Why should you choose Qibo?

- Publicly available as an open source project
- Community driven effort
- Specifically designed for self-hosted QPUs



Collaborators

Thanks for listening!

Questions time.