Quantum simulation and hardware control for HEP

Challenges and achievements in the NISQ era

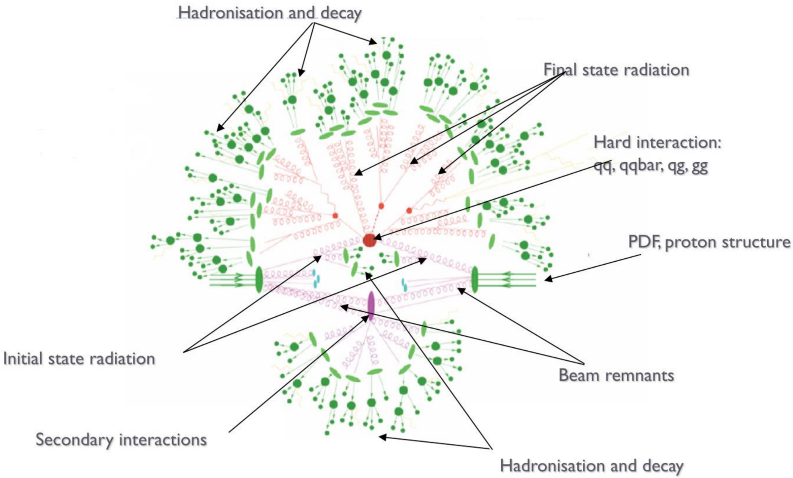
Stefano Carrazza, Università degli Studi di Milano

November 8th, 2023

Quantum observables for collider physics, GGI, Firenze

HEP challenges for LHC and future colliders

Monte Carlo simulation and data analysis are **intensive** and requires lots of **computing power**.



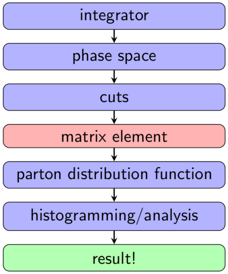
Parton-level Monte Carlo generators

Theoretical predictions in `hep-ph` are based on:

$$\sum_{a,b} \int_{x_{\min}}^{1} dx_1 dx_2 \, |\mathcal{M}_{ab}(\{p_n\})|^2 \, \mathcal{J}_m^n(\{p_n\}) \, f_a(x_1, Q^2) f_b(x_2, Q^2),$$

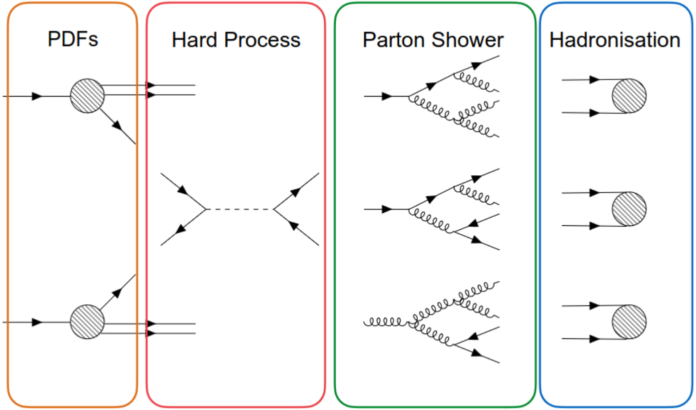
a multi-dimensional integral where:

- $|\mathcal{M}|$ is the matrix element,
- $f_i(x, Q^2)$ are Parton Distribution Functions (PDFs),
- $\{p_n\}$ phase space for n particles,
- $\mathcal{J}_m^n$ jet function for n particles to m.



$\Rightarrow$ **Procedure driven by the integration algorithm.**

Monte Carlo generator pipeline



R&D and adoption of new technologies in HEP

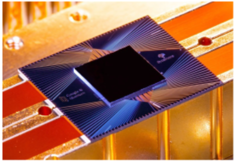
HEP is moving towards new technologies, in particular **hardware accelerators**:

CPU

GPU

FPGA/ASIC

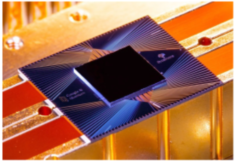
Quantum chip



Moving from **general purpose devices** $\Rightarrow$ **application specific**

R&D and adoption of new technologies in HEP

HEP is moving towards new technologies, in particular **hardware accelerators**:

CPU	GPU	FPGA/ASIC	Quantum chip
			

Moving from **general purpose devices** ⇒ **application specific**

Examples of **initiatives** and institutions involved:

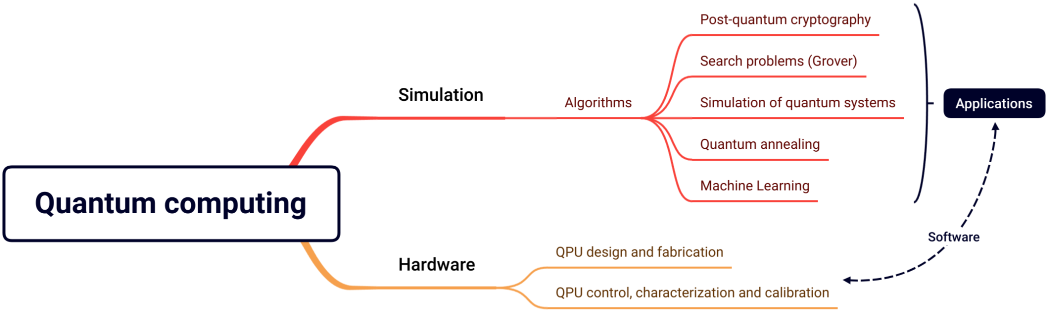
CERN | QUANTUM TECHNOLOGY INITIATIVE

INFN

UNIVERSITAS STUDIORUM MEDIOLANENSIS

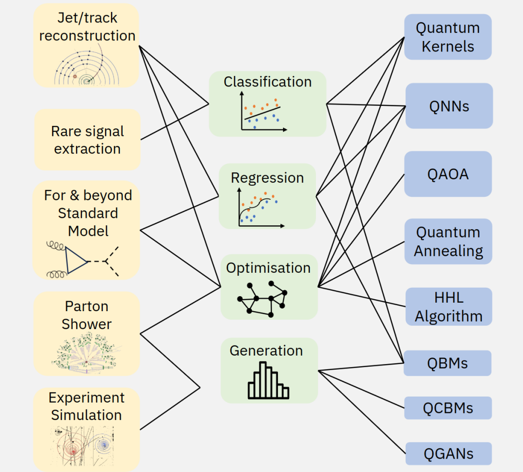
BIQUTE

Quantum Computing topics in HEP



The **HEP community** is testing **quantum computing algorithms** in topics related to:

hep-exp	hep-ph	quant-ph
Data analysis	Theoretical modelling	Software / Middleware



QC4HEP WG

Goal:

Replace **classical ML data analysis** methods with variational quantum computing (QML) and observe **advantage** with quantum computing methods.

How?

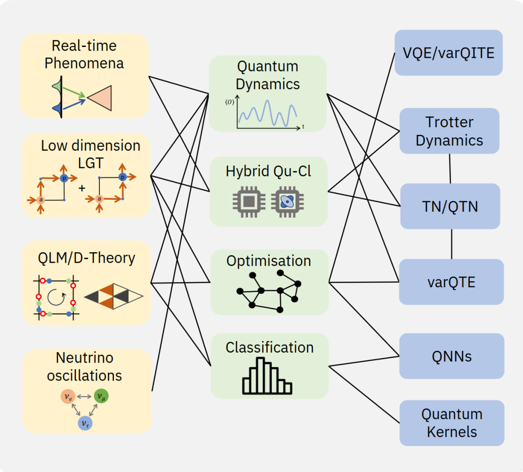
- Developing **variational models** using classical quantum simulation.
- Adapting problems and deploying strategies on **NISQ hardware**.

Goal:

Design **new algorithms** for QFT and Hadronic physics observables, identify **advantage** from quantum computing methods.

How?

- Designing **hybrid quantum-classical** methods using classical quantum simulation.
- Deploying **classical quantum simulation** techniques on HPC infrastructure.



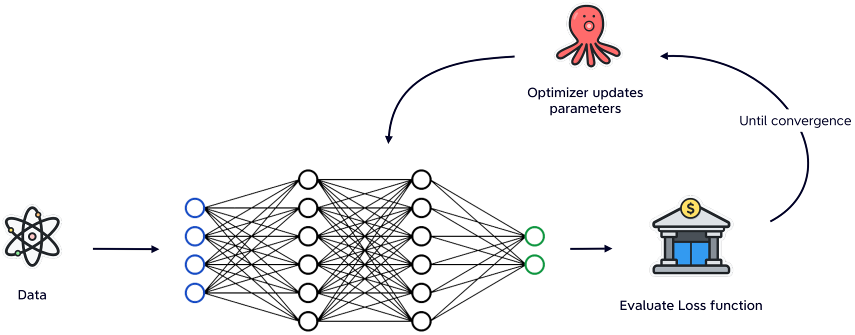
QC4HEP WG

Quantum machine learning

From classical Machine Learning to quantum

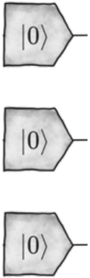
Classical **Machine Learning** solves statistical problems such as data generation, classification, regression, forecasting, etc.

- Aims to know some hidden law between two variables: $\boldsymbol{y} = f(\boldsymbol{x})$;
- Defines a parametric model with returns $\boldsymbol{y}_{\rm est} = f_{\rm est}(\boldsymbol{x}; \boldsymbol{\theta})$;
- Defines an optimizer, which task is to compute $\text{argmin}_{\boldsymbol{\theta}} \big[J(\boldsymbol{y}_{\rm meas}, \boldsymbol{y}_{\rm est}) \big]$.



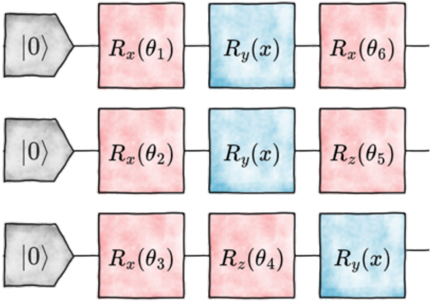
Parametric Quantum Circuits

- Classical bits are replaced by **qubits**: $|q\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$;



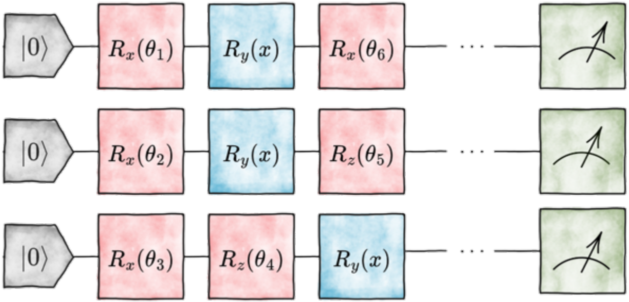
Parametric Quantum Circuits

- Classical bits are replaced by **qubits**: $|q\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$;
- The qubit state is modified by applying **gates** (unitary operators).
  Rotational gates $R_j(\theta) = e^{-i\theta\hat{\sigma}_j}$ are used to build parametric circuits $\mathcal{C}(\boldsymbol{\theta})$;



Parametric Quantum Circuits

- Classical bits are replaced by **qubits**: $|q\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$;
- The qubit state is modified by applying **gates** (unitary operators).
  Rotational gates $R_j(\theta) = e^{-i\theta\hat{\sigma}_j}$ are used to build parametric circuits $\mathcal{C}(\boldsymbol{\theta})$;
- Information is accessed calculating expected values $\mathrm{E}[\hat{O}\,]$ of target observables $\hat{O}$ on the state obtained executing $\mathcal{C}$.

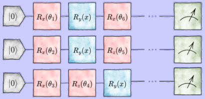


Quantum Machine Learning

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.

VQC execution



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

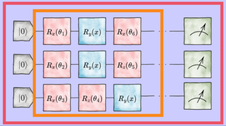
Quantum Machine Learning

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.

Expected values
$y_{est} \equiv \boxed{\langle q_f | B | q_f \rangle}$

VQC execution



Quantum Machine Learning

Optimizer $\mathcal{O}$
Hybrid-strategy

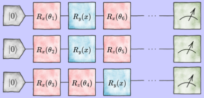
Machine Learning
$\mathcal{M}$: model;
$\mathcal{O}$: optimizer;
$\mathcal{J}$: loss function.
(x, y): data

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

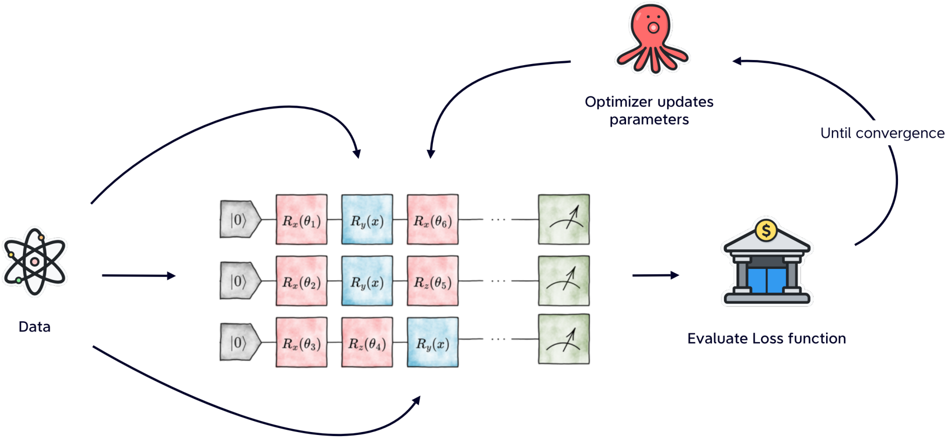
Expected values
$y_{est} \equiv \langle q_f | B | q_f \rangle$

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

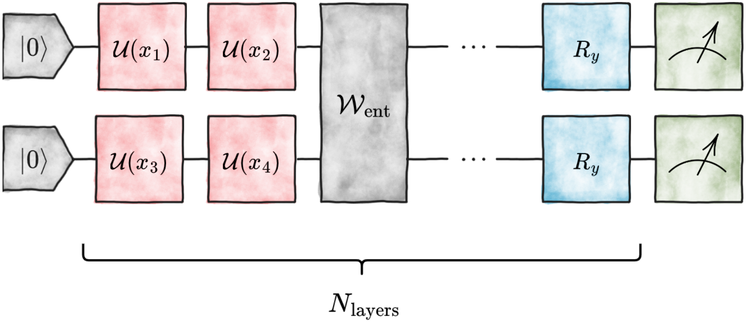
VQC execution



From ML to QML

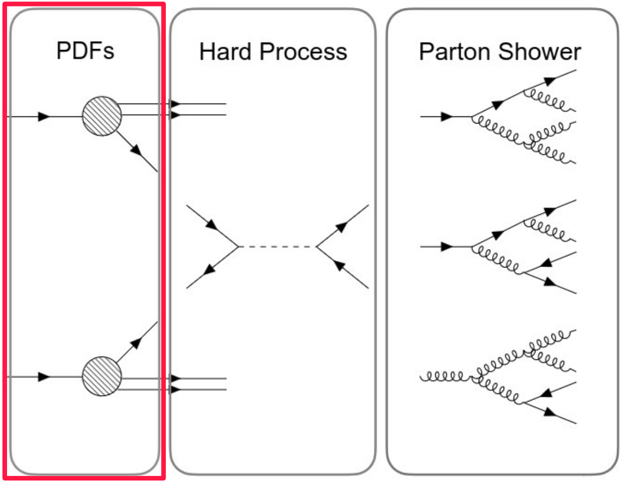


We define an uploading channel $U(x;\boldsymbol{\theta})$, and we repeat the uploading N times.



It has been proved this approach is equivalent to approximate a function with an N-term Fourier Series.

Example 1: Parton Distribution Functions

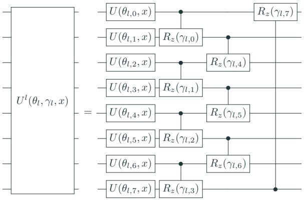


Parton distribution functions
(Machine Learning)

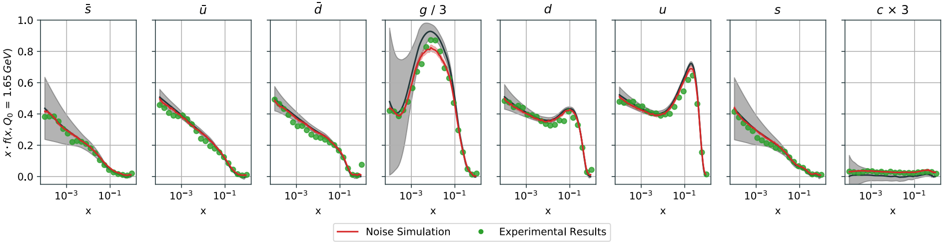
We parametrize **Parton Distribution Functions** with multi-qubit variational quantum circuits:

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 $z_i(\theta, x) = \langle \psi(\theta, x)|Z_i|\psi(\theta, x)\rangle$:

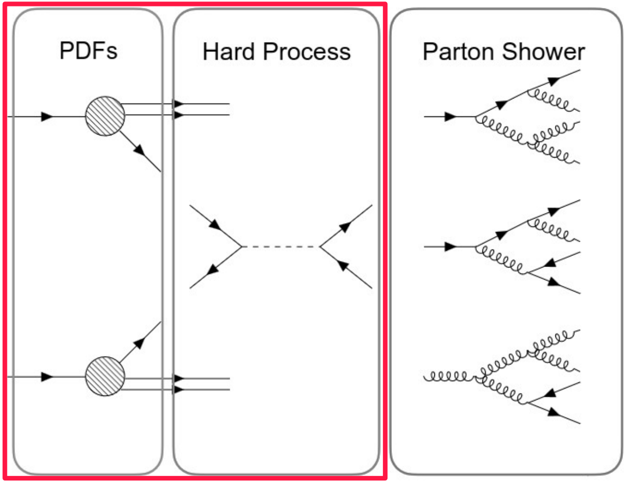
$$\mathrm{qPDF}_i(x, Q_0, \theta) = \frac{1 - z_i(\theta, x)}{1 + z_i(\theta, x)}.$$



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



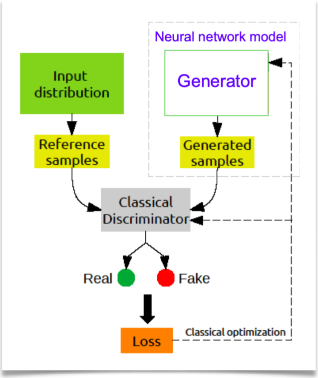
Example 2: Event generation



Event generation

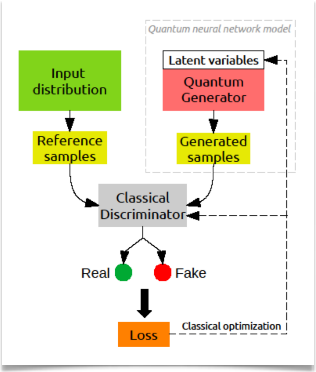
Train with a **small dataset**, use **unsupervised machine learning models** to learn the underlying distribution and generate for free a much larger dataset.

Classical setup:

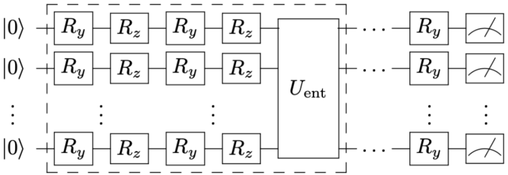


Only the generator becomes quantum

Hybrid quantum-classical setup:



Quantum generator: a series of quantum layers with rotation and entanglement gates



1 component = 1 qubit

$$\boldsymbol{x} = \left(-\langle\sigma_z^1\rangle, -\langle\sigma_z^2\rangle, \ldots, -\langle\sigma_z^n\rangle\right)$$

Style-based approach

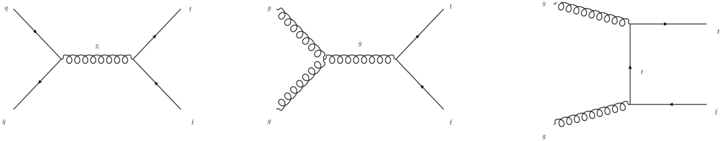
the noise is inserted in **every gate and not only in the initial quantum state**

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

Reminiscent of the reuploading scheme

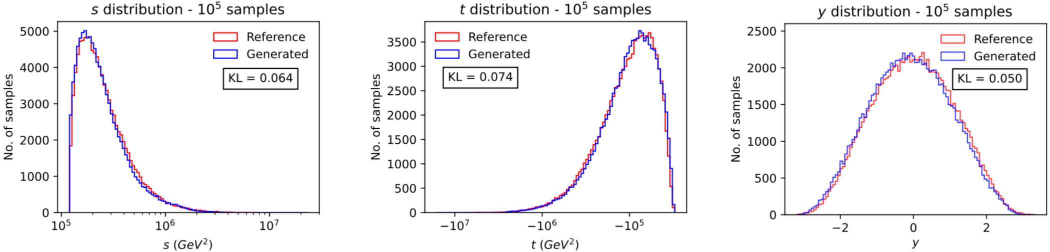
A. Pérez-Salinas, et al., *Quantum* **4**, 226 (2020)

Testing the style-qGAN with real data: proton-proton collision $pp \to t\bar{t}$



Training and reference samples for **Mandelstam variables** (s, t) and rapidity y generated with MadGraph5_aMC@NLO.

Simulation results: 3 qubits, 2 layers, 100 bins



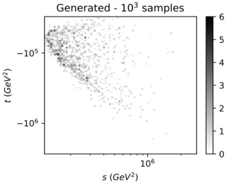
Testing different architectures

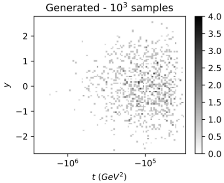
arXiv:2110.06933

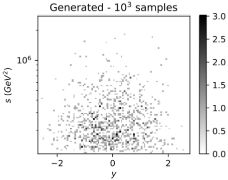
- Access constraints to *ionQ*: test limited to 1000 samples only

Very similar results:
implementation largely hardware-independent

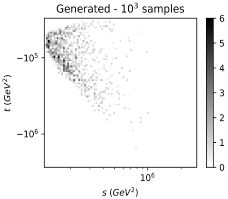
ionQ samples:

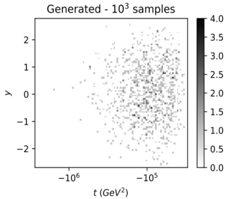


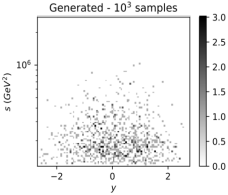




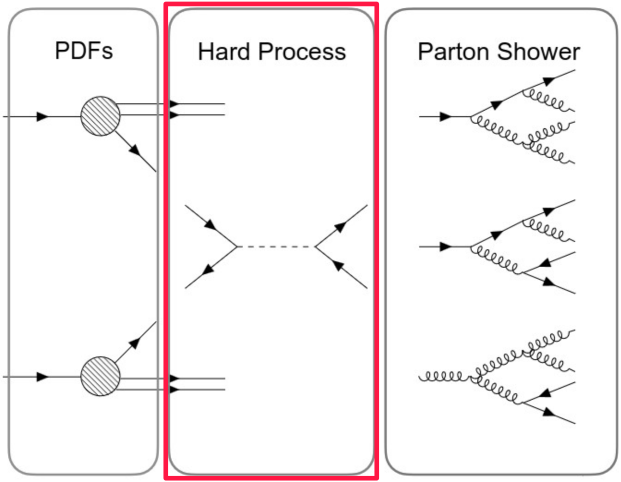
IBM Q samples:







Example 3: Monte Carlo Integration / Sampling

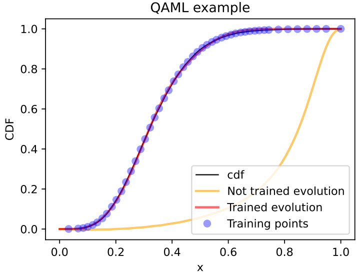


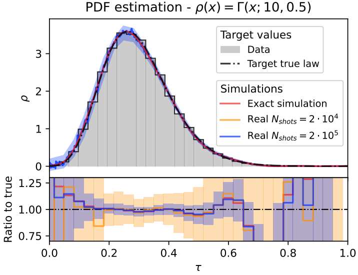
Monte Carlo Integration

Determining Probability Density Functions (PDF) by fitting the corresponding Cumulative Density Function (CDF) using an adiabatic QML ansatz.

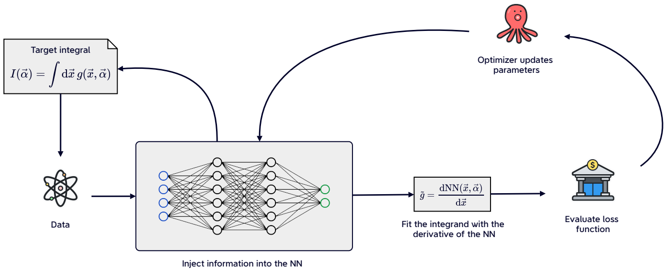
Algorithm's summary:

1. Optimize the parameters $\boldsymbol{\theta}$ using adiabatic evolution: $H_{\rm ad}(\tau;\boldsymbol{\theta}) = [1 - s(\tau;\boldsymbol{\theta})]\hat{X} + s(\tau;\boldsymbol{\theta})\hat{Z}$ in order to approximate some target CDF values;
2. Derivate from $H_{\rm ad}$ a circuit $\mathcal{C}(\tau;\boldsymbol{\theta})$ whose action on the ground state of $\hat{X}$ returns $|\psi(\tau)\rangle$;
3. The circuit at step 2 can be used to calculate the CDF;
4. Compute the PDF by derivating $\mathcal{C}$ with respect to τ using the Parameter Shift Rule.





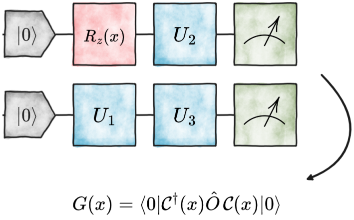
Multi-variable integrals using Neural Networks:

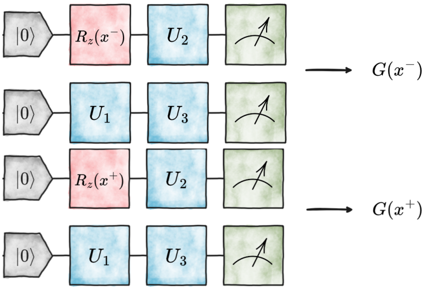


- both NN and dNN are models of the integral and integrand respectively;
- once trained, the NN can be called with any combination of data and parameters. Monte Carlo Integration (MCI), instead, has to be recomputed every time;
- in the INN is the integrand to be approximated, instead of the integral (as in MCI), swaps **variance** for approximation error.

Quantum inspiration - Parameter Shift Rule

arXiv:1811.11184





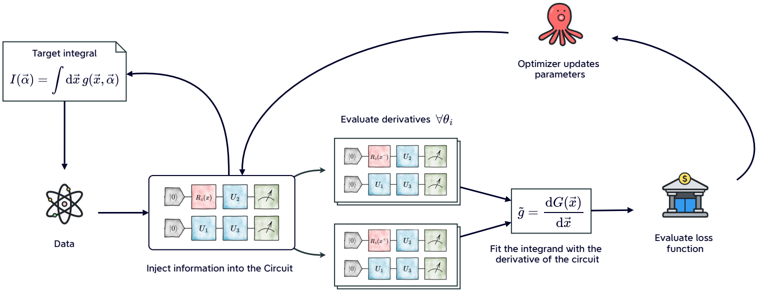
Considering the unitary $\mathcal{U}(x) = e^{-ixU}$ affected by one parameter x, if the hermitian generator U has at most two eigenvalues $\pm r$, an exact estimator of $\partial_x G$ is:

$$\partial_x G = r\big[G(x^+) - G(x^-)\big].$$

Where $x^\pm = x \pm s$ and, considering rotational gates, we have $s = \pi/2$ and $r = 1/2$.

At this point, we know that:

1. **variables** can be **injected** into a quantum circuit as rotational angles;
2. the same circuit architecture $\mathcal{C}$ can be used to compute **both** the estimator and its derivatives.

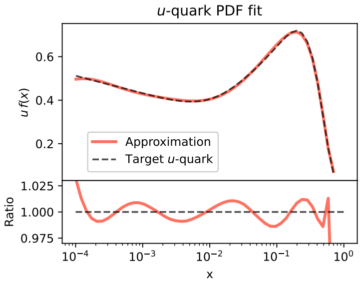


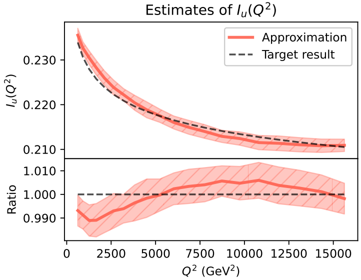
If independent variables, $\frac{dG(\boldsymbol{x})}{d\boldsymbol{x}}$ is obtained by summing all PSR contributions.

The u-quark PDF

Parameter	Value
$N_{x,\mathrm{train}}$	500
Q	1.67
N_{layers}	4
N_{params}	27
$\lvert I - \tilde{I} \rvert$	$1.2 \cdot 10^{-5}$
N_{shots}	Exact simulation
Optimizer	L-BFGS

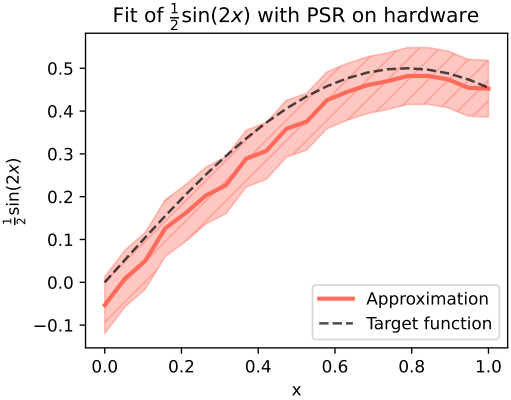
Parameter	Value
$(N_x, N_Q)_{\mathrm{train}}$	$(120, 100)$
$N_{Q,\mathrm{est}}$	20
N_{runs}	100
N_{layers}	4
N_{params}	36
$\lvert I - \tilde{I} \rvert$	$7.4 \cdot 10^{-5}$
N_{shots}	10^6
Optimizer	L-BFGS





We finally tackle a dummy target using a real superconducting qubit:

$$I = \int_0^1 \frac{1}{2}\sin(2x)\,\mathrm{d}x.$$



Parameter	Value
$N_{x,\mathrm{train}}$	50
$N_{x,\mathrm{est}}$	20
N_{runs}	10
N_{layers}	1
N_{params}	6
$\lvert I - \tilde{I}\rvert$	$2.8 \cdot 10^{-2}$
N_{shots}	$2 \cdot 10^3$
Optimizer	L-BFGS

Middleware challenges

Software challenges

How to **design** quantum algorithms and **deploy** on quantum hardware?

- Cloud-based quantum hardware:
  - Use tools provided by providers.
- Self-hosted quantum hardware:
  - Operate self-hosted quantum hardware.
  - Accommodate custom lab setups.
  - Bypass restrictions to execute an experiment.

Industry (cloud-based)

Software challenges

How to **design** quantum algorithms and **deploy** on quantum hardware?

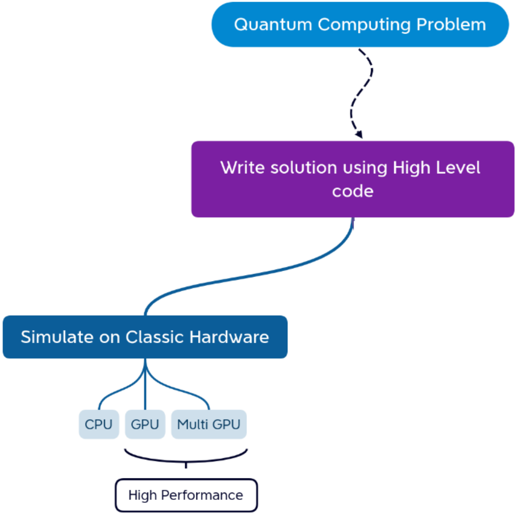
- Cloud-based quantum hardware:
  - Use tools provided by providers.
- Self-hosted quantum hardware:
  - Operate self-hosted quantum hardware.
  - Accommodate custom lab setups.
  - Bypass restrictions to execute an experiment.

Industry (cloud-based)

Example

Let's consider the PDF determination project, it requires two stages: **prototyping** and **deployment**.

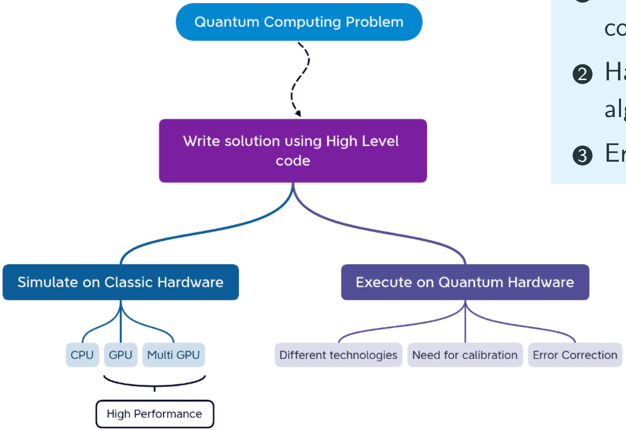
Stage 1: Prototyping models / algorithms



Prototyping

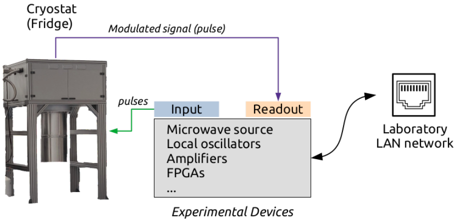
1. High-level quantum circuit programming language
2. Fast classical quantum simulation for model prototyping

Stage 2: Deployment on quantum hardware



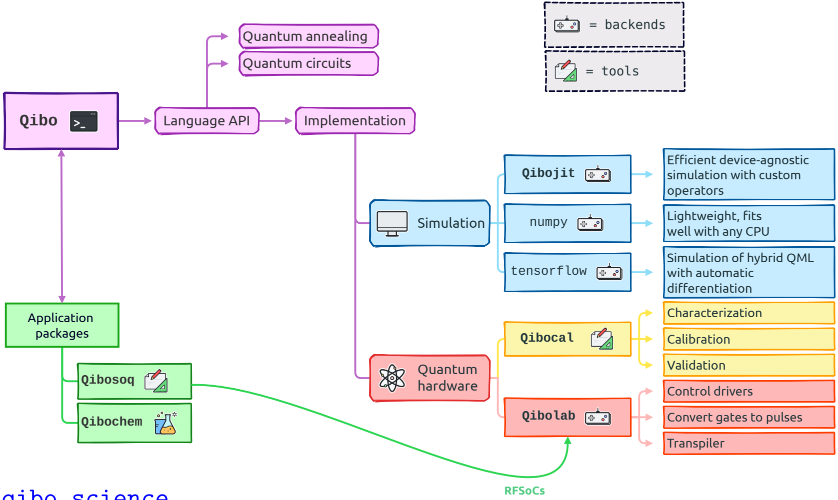
Deployment

1. Gates to microwave pulses sequence compilation (SC qubits)
2. Hardware compatible optimization algorithms
3. Error-mitigation algorithms



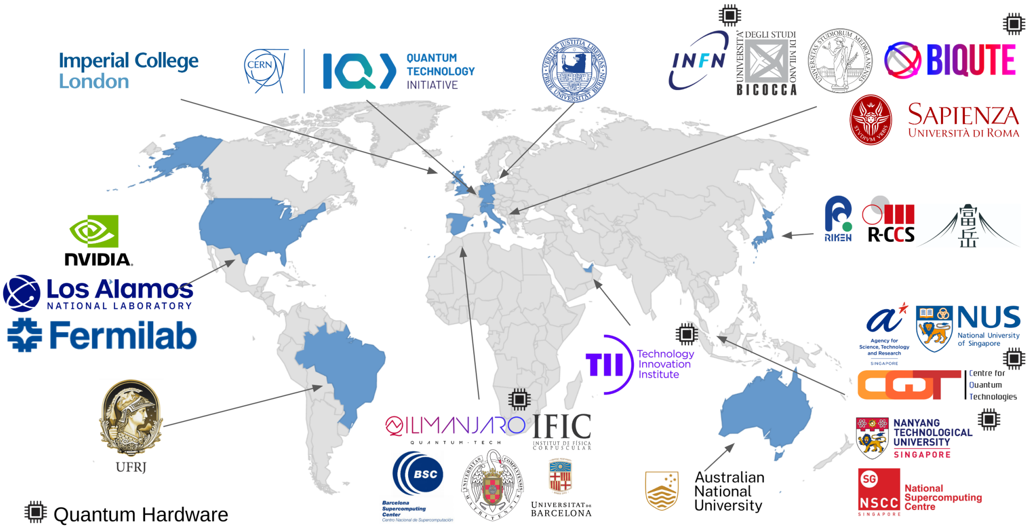
Introducing Qibo arXiv:2009.01845

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



https://qibo.science

Qibo Contributors (November 2023)



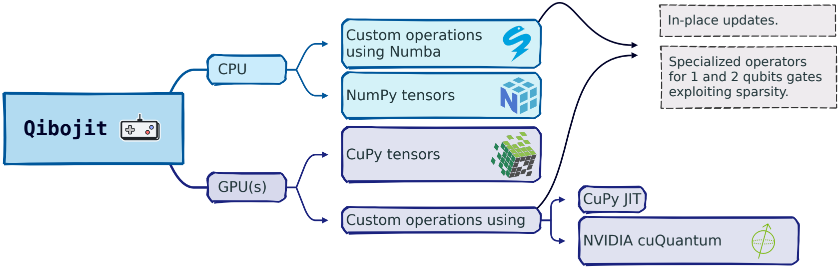
Laboratory	Country	Technology	Qubits
INFN	Italy	Superconducting	1
UNIMIB	Italy	Superconducting	1
TII	UAE	Superconducting	1, 2, 5, 25
Qilimanjaro	Spain	Superconducting	1 and 2
CQT	Singapore	SC and trapped ion	10

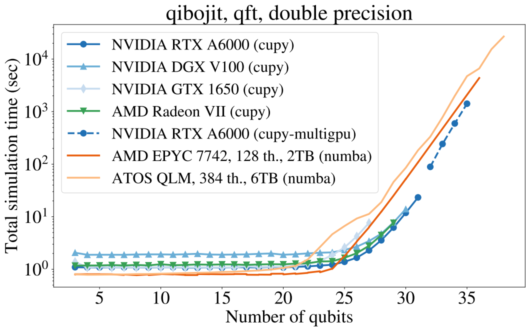
State vector simulation solves:

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

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

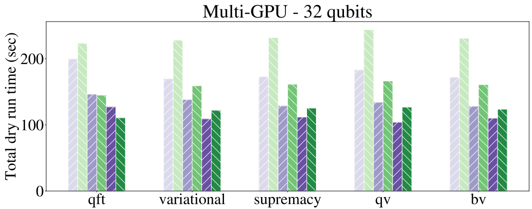
Qibo uses just-in-time technology and hardware acceleration:





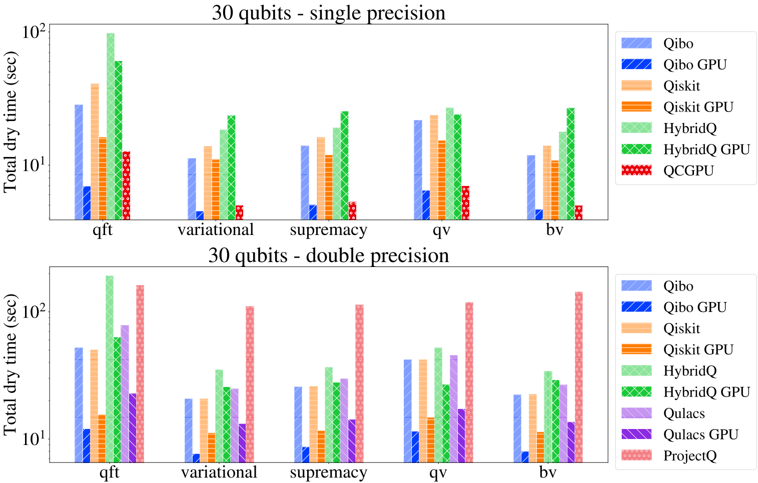
Major features:

- Supports CPU, GPU and multi-GPU.
- NVIDIA and AMD GPUs support.
- Reduced memory footprint.



Benchmark library: `https://github.com/qiboteam/qibojit-benchmarks`

Benchmark library: `https://github.com/qiboteam/qibojit-benchmarks`



The Tensor Network approach

Advantages

- Efficient on large systems > 40 qubits.
- Potential tool to solve gauge theories (quantum annealing / Hamiltonian computing).

Disadvantages

- How many qubits do we really need?
- How to validate results?
- Limited to few observables?
- Software is HPC-infrastructure dependent.

On-going: QiboTN module for Tensor Networks simulation on GPUs.

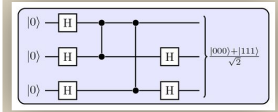
How to control qubits?

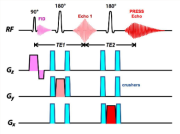
arXiv:2308.06313

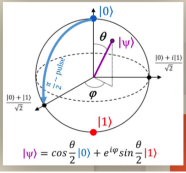
Remote client | On board server

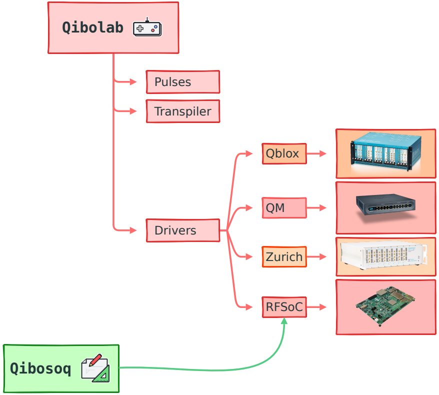
Circuit definition → **Transpilation** → **Hardware deployment** → **Experiment execution**

Qibo | Qibolab | Qibosoq



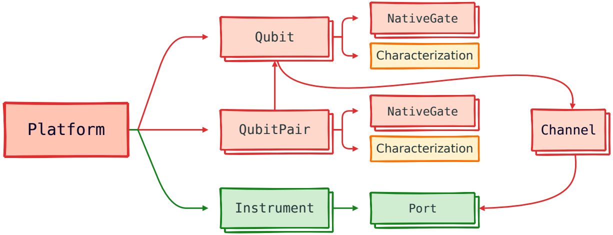






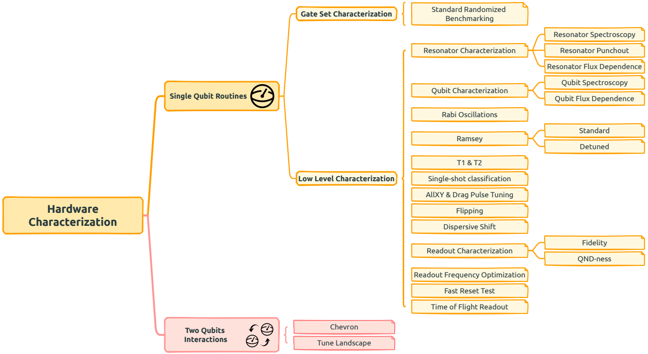
Major features:

- Pulse and pulse sequence API.
- Drivers for lab instruments, including AWGs and FPGAs.
- Hardware sweeps for faster execution of calibration routines.
- Transpilers from arbitrary circuits to pulses.



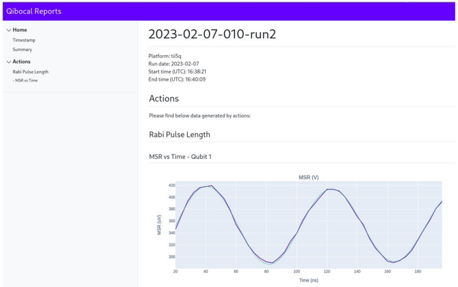
Qubit characterization and calibration

arXiv:2308.06313

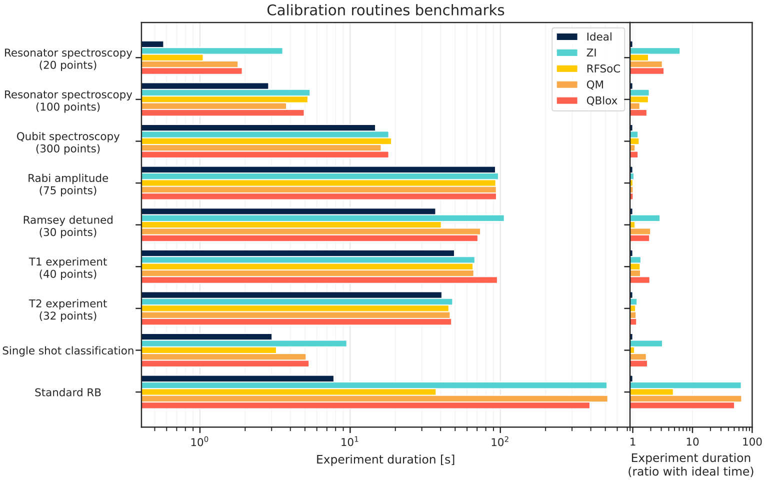


Major features:

Calibration of single and multi-qubit platforms are possible using **Qibo**.



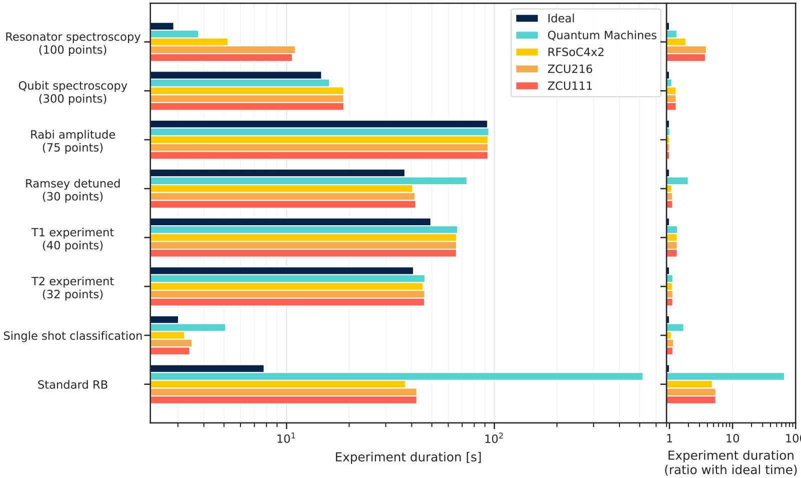
We compare the ideal pulse sequence execution performance to instruments execution duration.



Zurich Instruments (ZI), Quantum Machines (QM), QBlox and RFSoC FPGA (Qibosoq+QICK).

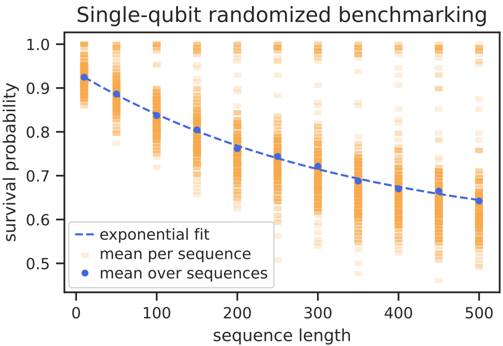
Benchmarking instruments performance

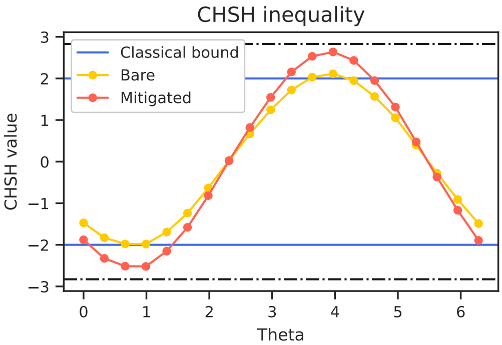
arXiv:2308.06313



FPGA RFSoC vs commercial products.

Examples of results executed on quantum hardware (single-qubit) after calibration with Qibo:





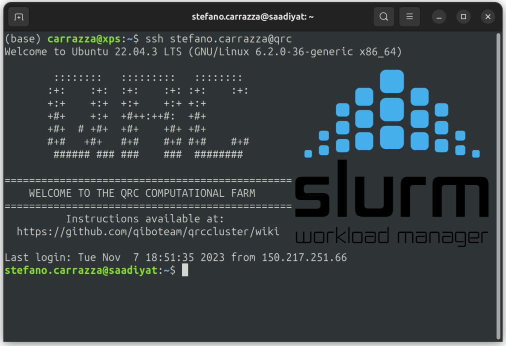
QPU status monitoring tool

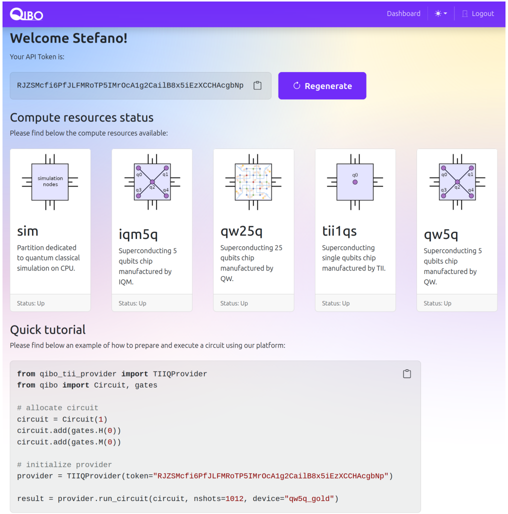
Qibo also provides monitoring tools for QPU control and alarms for calibration through Grafana and other custom tools.

Remote connection

Qibo provides remote access plugins for:

- Advanced users: using slurm plugin.
- Basic users: through a web-interface and HTTP rest API.





Remote connection

jupyter test Last Checkpoint: yesterday

File Edit View Run Kernel Settings Help — Trusted

Code — Open in... — Python 3 (ipykernel)

[1]:
```python
from qibo_tii_provider import TIIProvider

from qibo import Circuit, gates

# allocate circuit
circuit = Circuit(1)
circuit.add(gates.H(0))
circuit.add(gates.M(0))

# initialize provider
provider = TIIProvider(token="AMshDIUQnjuqBHnQ6TYI1WdJBCzl4tJI2TIRrE2f3va5llLwBO")

result = provider.run_circuit(circuit, nshots=1012, device="tii1q_b1")
```

```
Post new circuit on the server
Job posted on server with pid e293377577004aca913ae25356085f26
Check results every 2 seconds ...
```

[2]:
```python
result.frequencies()
```

[2]:
```
Counter({'0': 516, '1': 496})
```

A full-stack case study

1. High-Level API (Qibo)

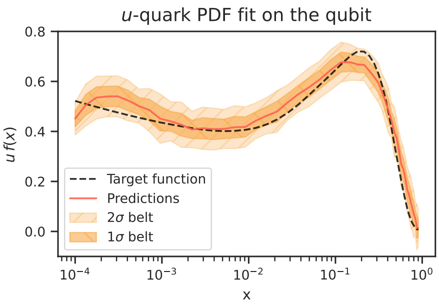
- Define model prototype.
- Implement training loop.
- Perform training using simulation.

2. Calibration (Qibocal)

- Calibrate qubit.
- Generate platform configuration.

3. Execution (Qibolab)

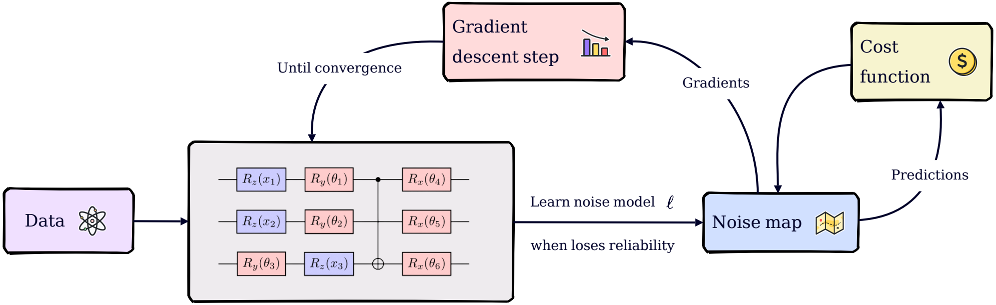
- Allocate calibrated platform.
- Compile and transpile circuit.
- Execute model and return results.



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

Real-time error mitigation in QML trainings

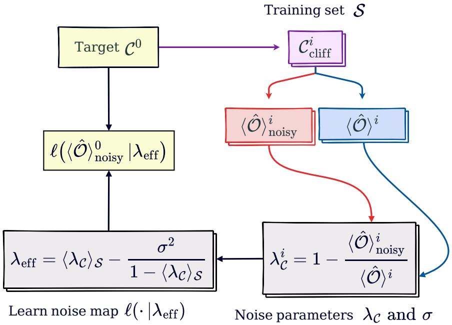
We define a Real-Time Quantum Error Mitigation (RTQEM) procedure.



- consider a Variational Quantum Algorithm trained with gradient descent;
- learn the noise map ℓ every time is needed over the procedure;
- use ℓ to clean up both predictions and gradients.

Learning the noise model arXiv:2112.06255

We use the Importance Clifford Sampling (ICS) procedure to learn the noise map ℓ.



Algorithm 1: RTQEM

`# ---------- Initialization`

Set $N_{\text{update}}, N_{epoch}, k = 0$;
Initialize VQC parameters $\boldsymbol{\theta}_k$, noise map ℓ ;
Define target noiseless expectation value z ;
Define metric $D(z, \ell(z))$ to check ℓ reliability;

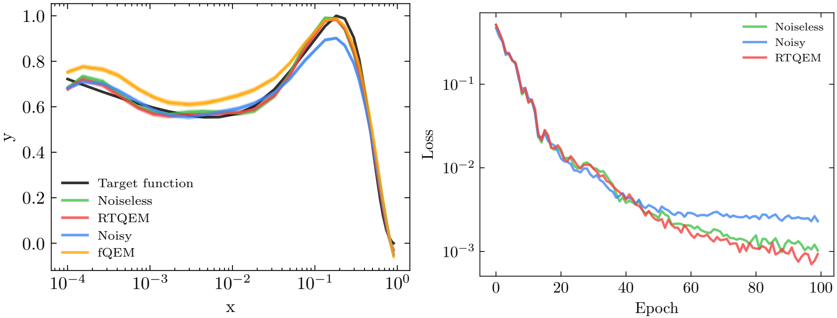
`# ---------- Execution`

for $k < N_{\text{epochs}}$ **do**
  if $D(z, \ell(z)) > \varepsilon_\ell$ **then**
    learn ℓ_k;
    $\ell \leftarrow \ell_k$;
  end
  compute $\ell(\boldsymbol{y}_{\text{est}})$;
  calculate $J\left[\ell(\boldsymbol{y}_{\text{est}}), \boldsymbol{y}_{\text{meas}}\right]$;
  for $p \in \boldsymbol{\theta_k}$ **do**
    compute $\ell(\partial_p J)$ via PSR;
  end
  $\boldsymbol{\theta}_{k+1} \leftarrow \text{Adam}(\boldsymbol{\theta}_k)$;
end

1. sample a training set of Clifford circuits $\mathcal{S}$ on top of a target $\mathcal{C}^0$;
2. process them so that their expectation values on Pauli strings is $+1$ or -1;
3. extract mitigation parameter λ_{eff} comparing $\langle\hat{\mathcal{O}}\rangle_{\text{noisy}}$ and $\langle\hat{\mathcal{O}}\rangle$;
4. build $\ell \equiv \ell(\cdot|\lambda_{\text{eff}})$ following the Phenomenological-Error-Model Inspired (PEMI) protocol.

One dimensional HEP target: the u-quark PDF

Parameter	N_{train}	N_{params}	N_{shots}	$\text{MSE}_{\text{rtqem}}$	$\text{MSE}_{\text{nomit}}$	Noise
Value	30	16	10^4	0.008	0.018	local Pauli

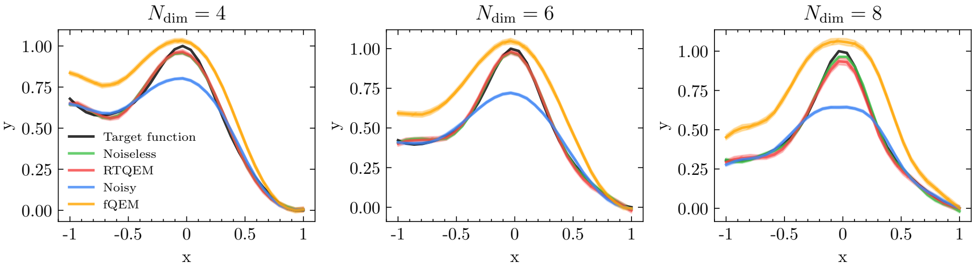


1. thanks to the RTQEM procedure, we reach a good minimum of the cost function;
2. the QEM is not effective is applied to a corrupted scenario (orange curve).

Multidimensional target

We tackle a multi-dimensional target computing predictions as expected value of a $Z^{\otimes N_{\rm dim}}$ after executing an $N_{\rm dim}$ circuit.

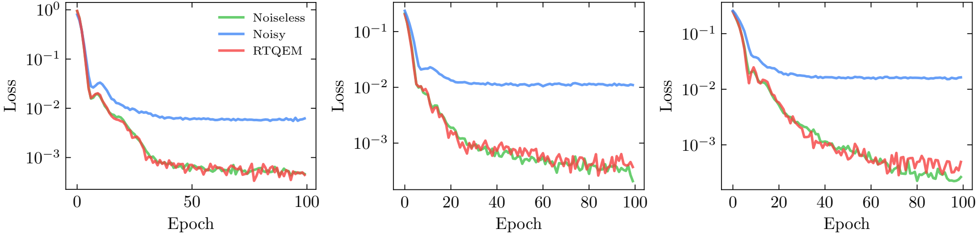
Job ID	$N_{\rm train}$	$N_{\rm params}$	$N_{\rm shots}$	MSE$_{\rm rtqem}$	MSE$_{\rm nomit}$	Noise
$N_{\rm dim} = 4$	30	48	10^4	0.003	0.043	local Pauli
$N_{\rm dim} = 6$	30	72	10^4	0.002	0.083	local Pauli
$N_{\rm dim} = 8$	30	96	10^4	0.004	0.118	local Pauli



Multidimensional target

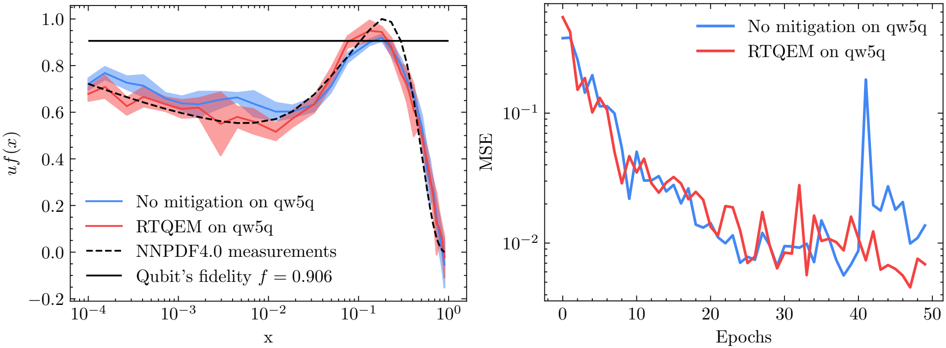
We tackle a multi-dimensional target computing predictions as expected value of a $Z^{\otimes N_{\text{dim}}}$ after executing an N_{dim} circuit.

Job ID	N_{train}	N_{params}	N_{shots}	$\text{MSE}_{\text{rtqem}}$	$\text{MSE}_{\text{nomit}}$	Noise
$N_{\text{dim}} = 4$	30	48	10^4	0.003	0.043	local Pauli
$N_{\text{dim}} = 6$	30	72	10^4	0.002	0.083	local Pauli
$N_{\text{dim}} = 8$	30	96	10^4	0.004	0.118	local Pauli



RTQEM on a superconducting qubit

Parameter	N_{train}	N_{params}	N_{shots}	$\text{MSE}^{\text{rtqem}}_{\text{best}}$	$\text{MSE}^{\text{unmit}}_{\text{best}}$	Noise
Value	15	16	500	0.0042	0.0055	real noise

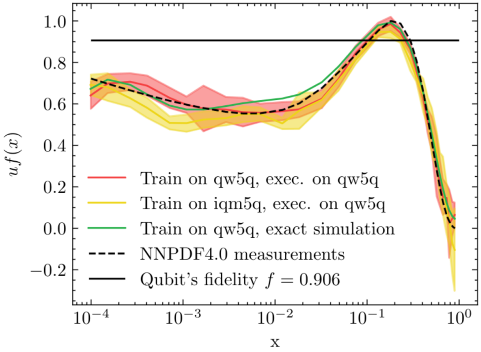


RTQEM allows exceeding the natural bound imposed by noise.

Can RTQEM generalise?

We perform a longer training on two different devices, same initial conditions and 100 epochs.

- `iqm5q` by IQM controlled using Zurich Instruments;
- `qw5q` by QuantWare controlled using Qblox.



Train.	Epochs	Pred.	Config.	MSE
`qw5q`	50	`qw5q`	noisy	0.0055
`qw5q`	50	`qw5q`	RTQEM	0.0042
`qw5q`	100	`qw5q`	RTQEM	0.0013
`iqm5q`	100	`qw5q`	RTQEM	0.0037
`qw5q`	100	`sim`	RTQEM	0.0016

All the hardware results are obtained deploying $\boldsymbol{\theta}_{\rm best}$ on `iqm5q`.

Outlook

Outlook

We have observed a great set of interesting **proof-of-concept** applications in HEP.

For the future:

- Improve results quality, moving from **prototype to production**.
- Mitigate hardware noise by implementing **real-time error mitigation** techniques.
- Provide **software tools** for further enhancement of quantum technologies.
- Enhance **calibration, characterization and validation** techniques.
- Codevelop **quantum hardware and instruments** for application specific tasks.