Quantum computers for particle theory

Challenges and achievements in the NISQ era

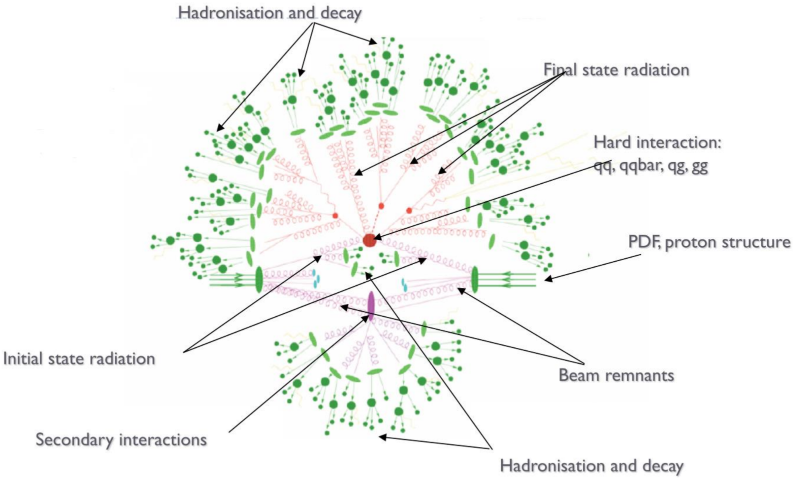
Stefano Carrazza, Università degli Studi di Milano

March 14th, 2024

ACAT2024, Stony Brook

HEP challenges for LHC and future colliders

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



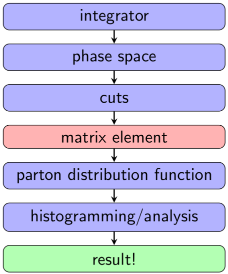
Parton-level Monte Carlo generators

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

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

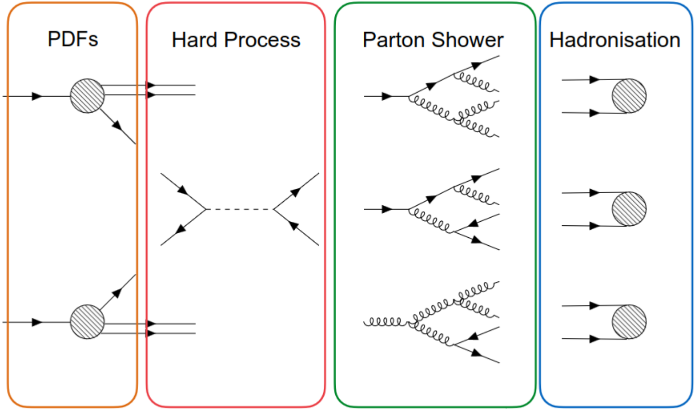
a multi-dimensional integral where:

- $|\mathcal{M}|$ 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

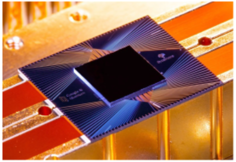
HEP is moving towards new technologies, in particular **hardware accelerators**:

CPU

GPU

FPGA/ASIC

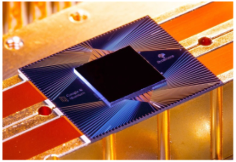
Quantum chip



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**:

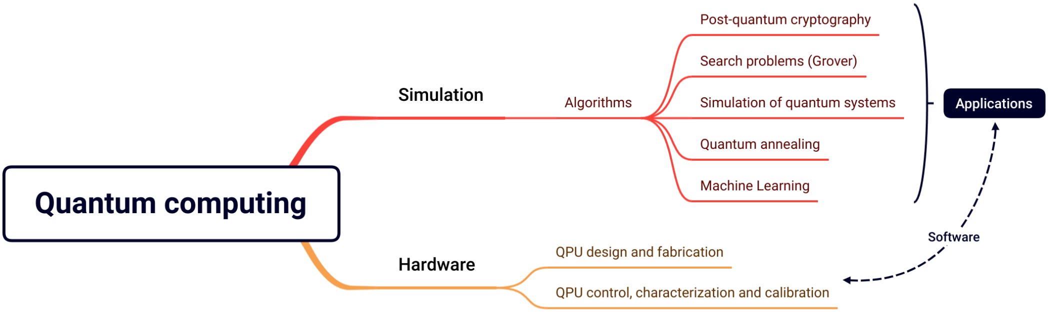
CPU	GPU	FPGA/ASIC	Quantum chip
			

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

Examples of **initiatives** and institutions involved:

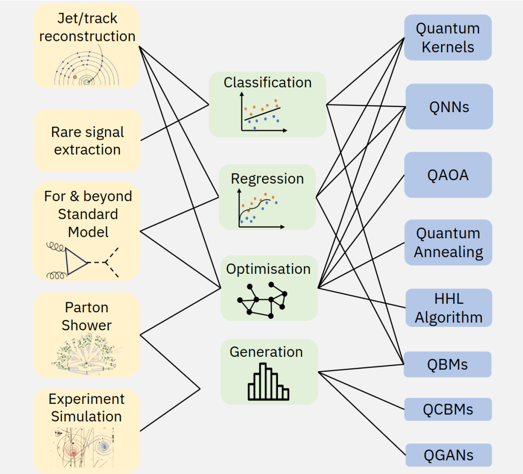
CERN | QUANTUM TECHNOLOGY INITIATIVE INFN BIQUTE

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?

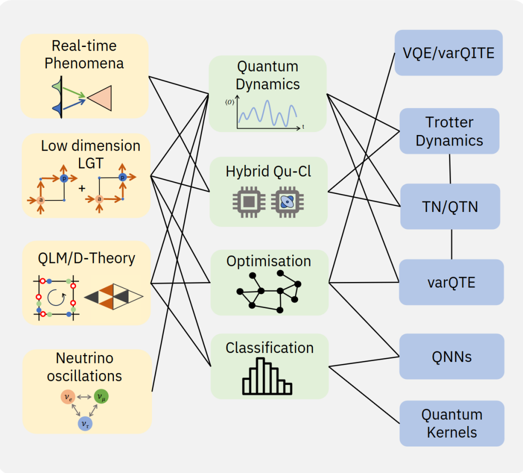
- Developing **variational models** using classical quantum simulation.
- Adapting problems and deploying strategies on **NISQ hardware**.

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.



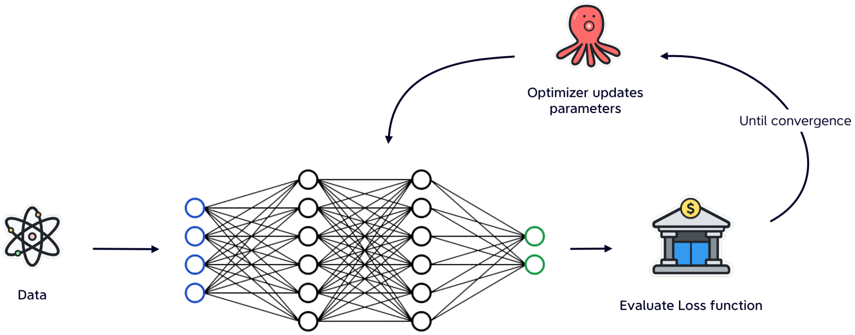
QC4HEP WG

Quantum machine learning

From classical Machine Learning to quantum

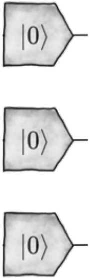
Classical **Machine Learning** solves statistical problems such as data generation, classification, regression, forecasting, etc.

- 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 $\text{argmin}_{\boldsymbol{\theta}}\big[J(\boldsymbol{y}_{\rm meas}, \boldsymbol{y}_{\rm est})\big]$.



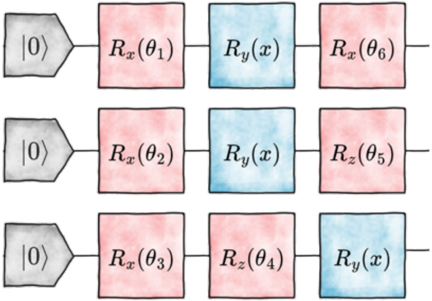
Parametric Quantum Circuits

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



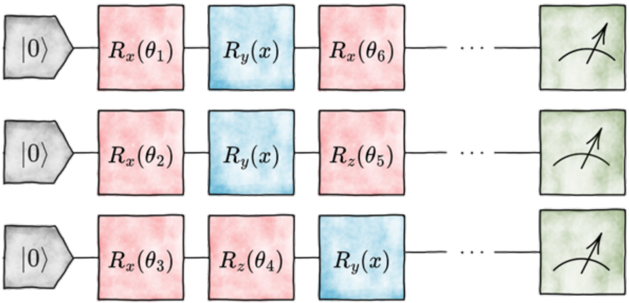
Parametric Quantum Circuits

- 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 $\mathcal{C}$.



Machine Learning

$\mathcal{M}$: model;

$\mathcal{O}$: optimizer;

$\mathcal{J}$: loss function.

(x, y): data

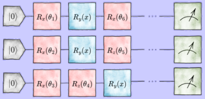
Quantum Computation

$\mathcal{Q}$: qubits;

$\mathcal{S}$: superposition;

$\mathcal{E}$: entanglement.

VQC execution



Expected values

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

Machine Learning

$\mathcal{M}$: model;

$\mathcal{O}$: optimizer;

$\mathcal{J}$: loss function.

(x, y): data

Quantum Computation

$\mathcal{Q}$: qubits;

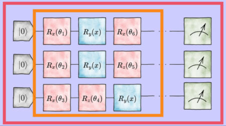
$\mathcal{S}$: superposition;

$\mathcal{E}$: entanglement.

Expected values

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

VQC execution



Quantum Machine Learning

Optimizer $\mathcal{O}$
Hybrid-strategy

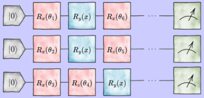
Machine Learning
$\mathcal{M}$: model;
$\mathcal{O}$: optimizer;
$\mathcal{J}$: loss function.
(x, y): data

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

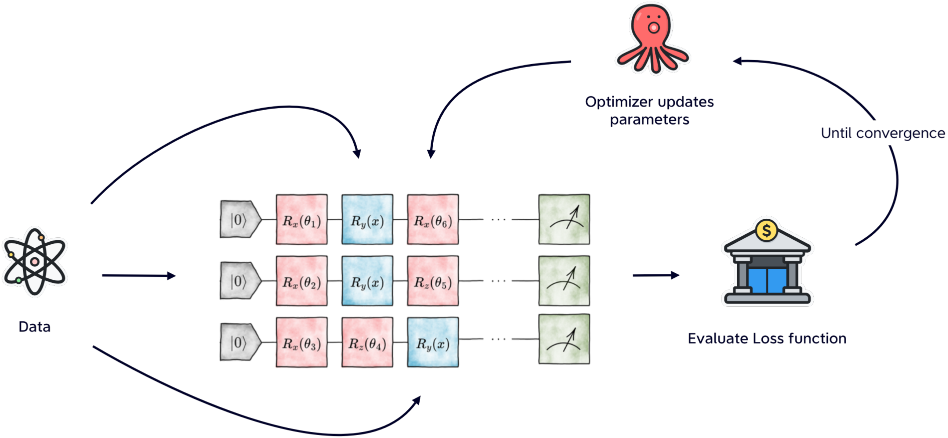
Expected values
$y_{est} \equiv \langle q_f | B | q_f \rangle$

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

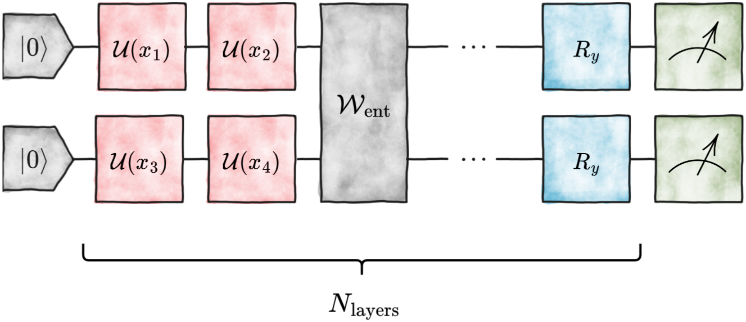
VQC execution



From ML to QML

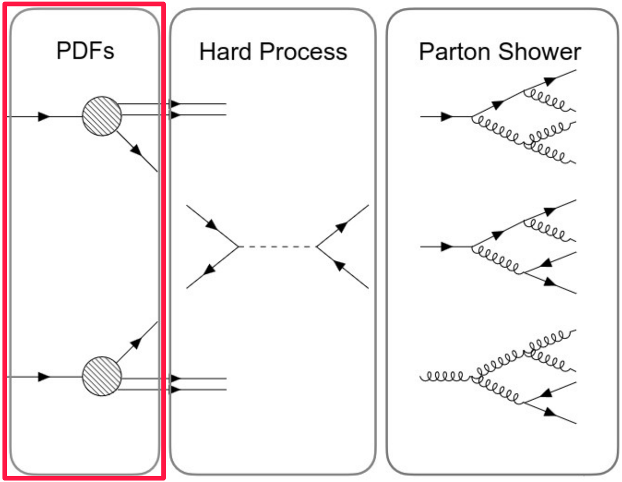


We define an uploading channel $U(x;\boldsymbol{\theta})$, and we repeat the uploading N times.



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)

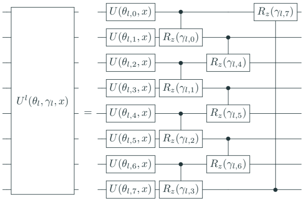
Determination of parton distribution functions

arXiv:2011.13934

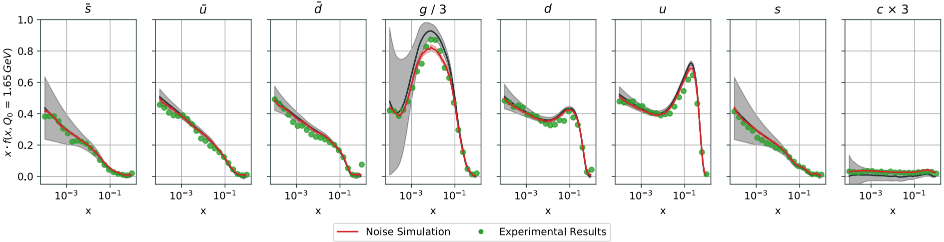
We parametrize **Parton Distribution Functions** with multi-qubit variational quantum circuits:

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

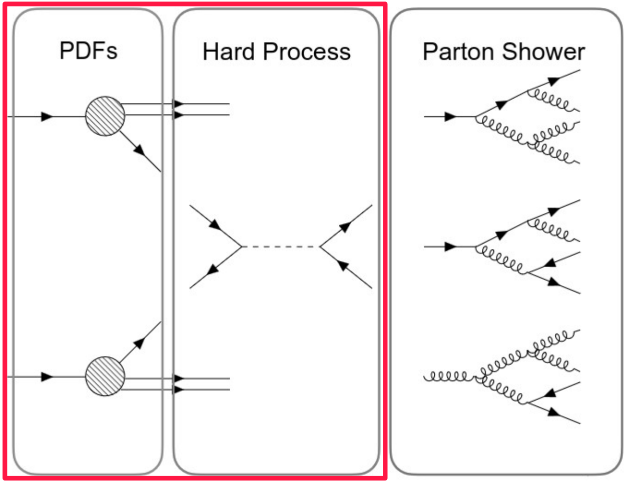
$$\text{qPDF}_i(x, Q_0, \theta) = \frac{1 - z_i(\theta, x)}{1 + z_i(\theta, x)}.$$



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



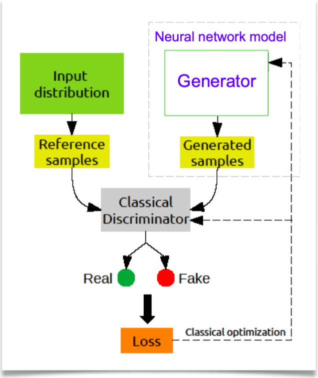
Example 2: Event generation



Event generation

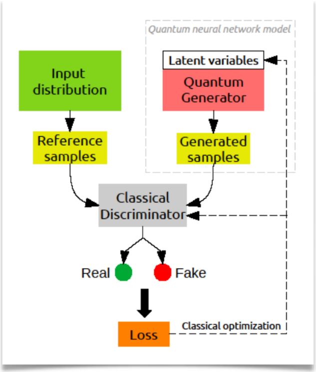
Train with a **small dataset**, use **unsupervised machine learning models** to learn the underlying distribution and generate for free a much larger dataset.

Classical setup:

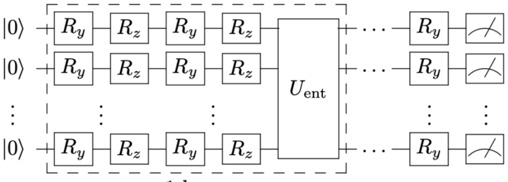


Only the generator becomes quantum

Hybrid quantum-classical setup:



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



1 component = 1 qubit

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

Style-based approach

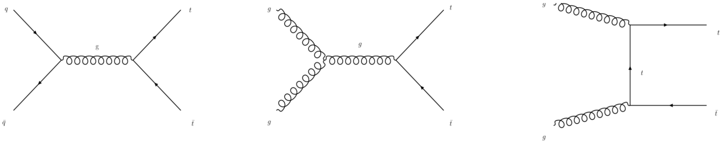
the noise is inserted in **every gate and not only in the initial quantum state**

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

Reminiscent of the reuploading scheme

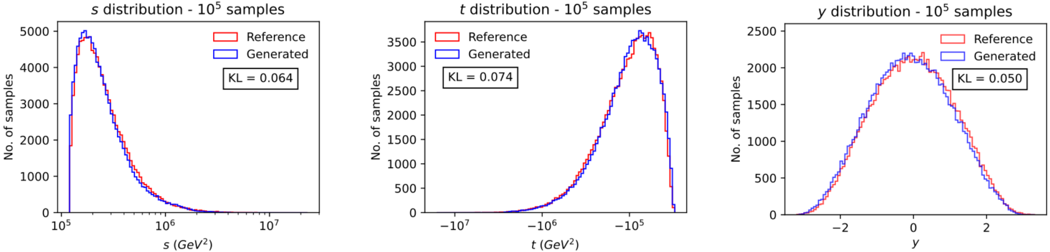
A. Pérez-Salinas, et al., *Quantum* **4**, 226 (2020)

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.

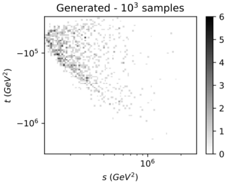
Simulation results: 3 qubits, 2 layers, 100 bins

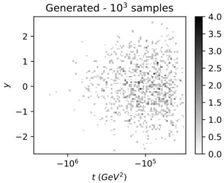


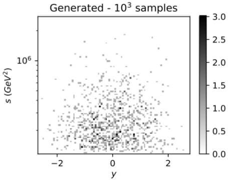
- Access constraints to *ionQ*: test limited to 1000 samples only

Very similar results:
implementation largely hardware-independent

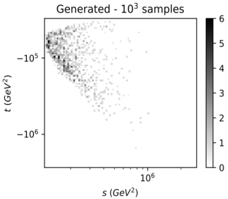
ionQ samples:

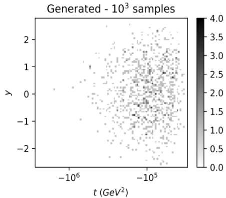


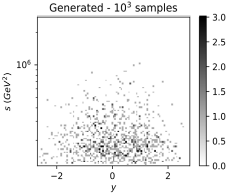




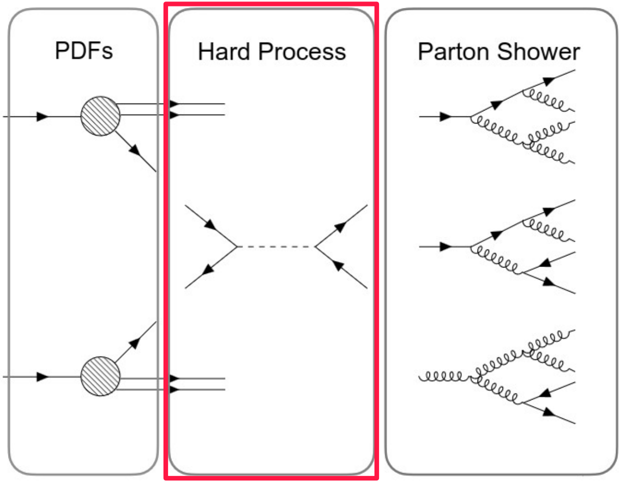
IBM Q samples:





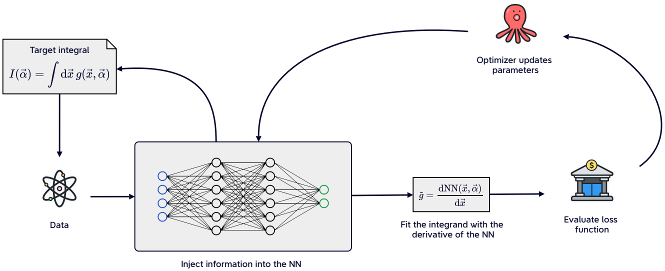


Example 3: Monte Carlo Integration / Sampling

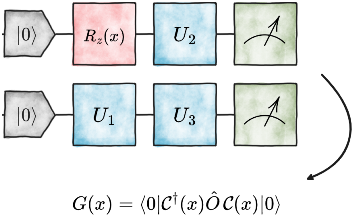


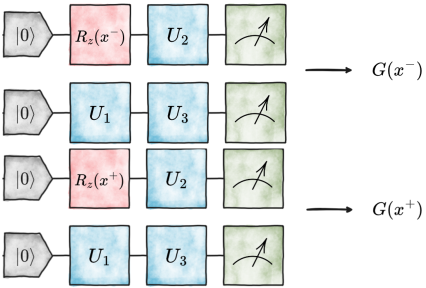
Monte Carlo Integration

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.





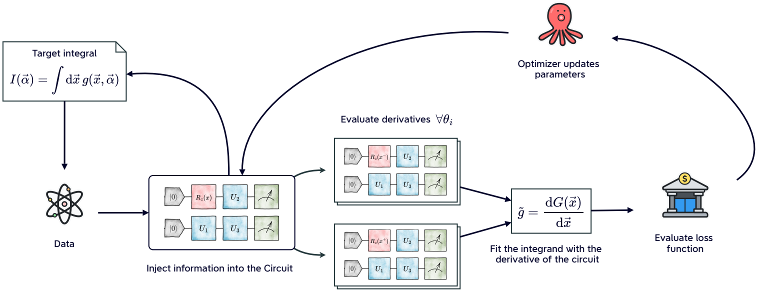
Considering the unitary $\mathcal{U}(x) = e^{-ixU}$ affected by one parameter x, if the hermitian generator U has at most two eigenvalues $\pm r$, an exact estimator of $\partial_x G$ is:

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

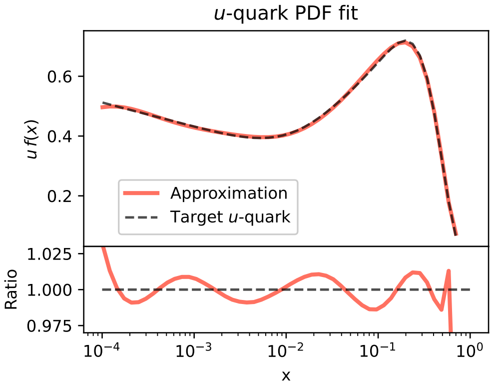
Where $x^{\pm} = x \pm s$ and, considering rotational gates, we have $s = \pi/2$ and $r = 1/2$.

At this point, we know that:

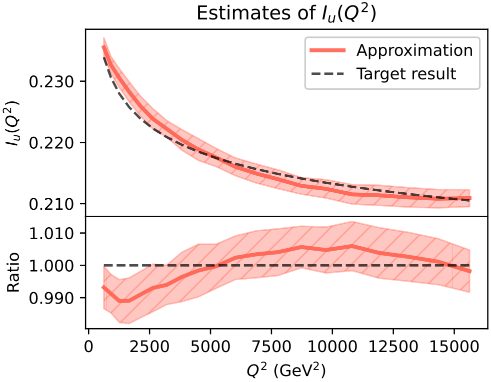
1. **variables** can be **injected** into a quantum circuit as rotational angles;
2. the same circuit architecture $\mathcal{C}$ can be used to compute **both** the estimator and its derivatives.



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



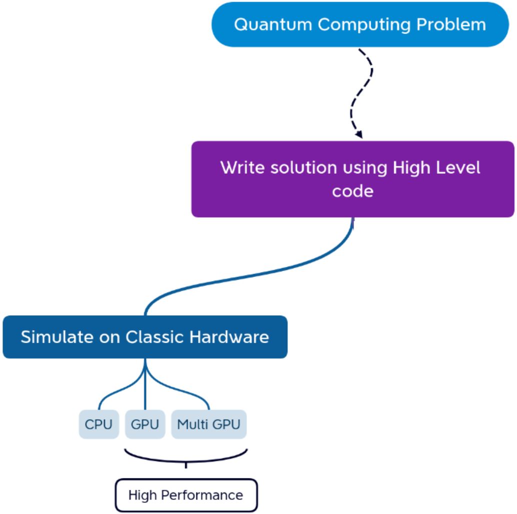
u-quark PDF fit
u f(x)
0.6
0.4
0.2
Approximation
Target u-quark
Ratio
1.025
1.000
0.975
10^-4 10^-3 10^-2 10^-1 10^0
x



Estimates of I_u(Q^2)
I_u(Q^2)
0.230
0.220
0.210
Approximation
Target result
Ratio
1.010
1.000
0.990
0 2500 5000 7500 10000 12500 15000
Q^2 (GeV^2)

Middleware challenges

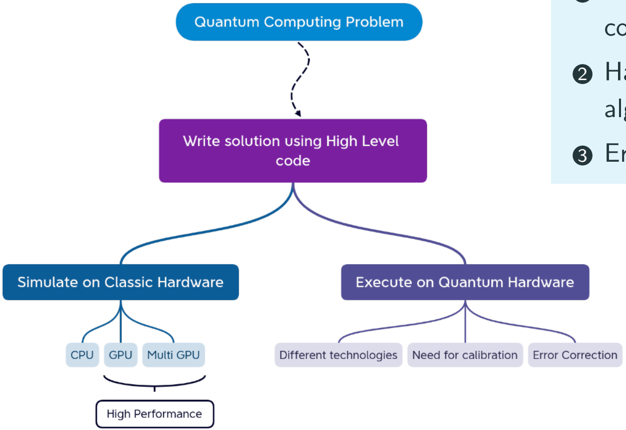
Stage 1: Prototyping models / algorithms



Prototyping

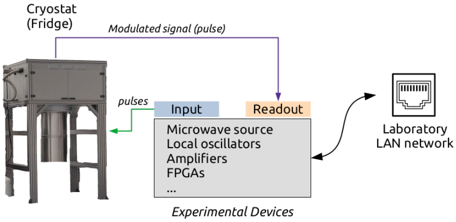
1. High-level quantum circuit programming language
2. Fast classical quantum simulation for model prototyping

Stage 2: Deployment on quantum hardware

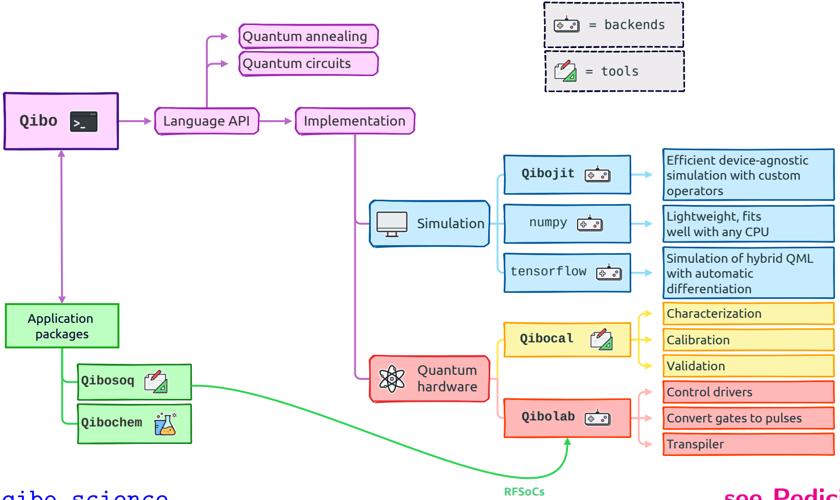


Deployment

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



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



https://qibo.science

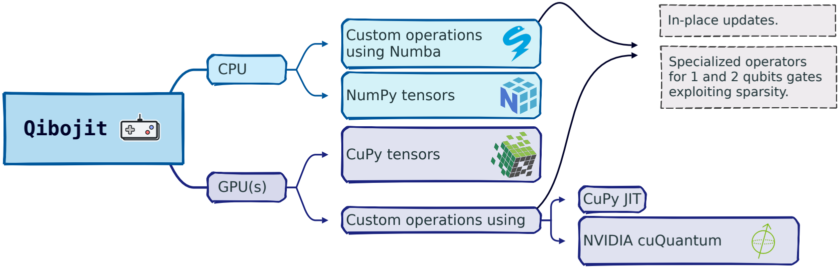
see Pedicillo's talk

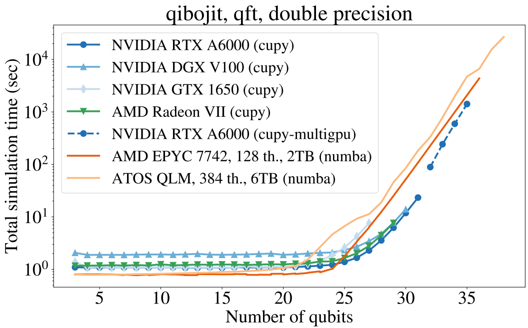
State vector simulation solves:

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

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

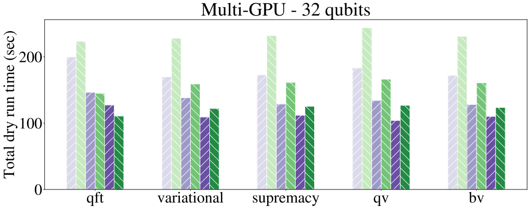
Qibo uses just-in-time technology and hardware acceleration:





Major features:

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

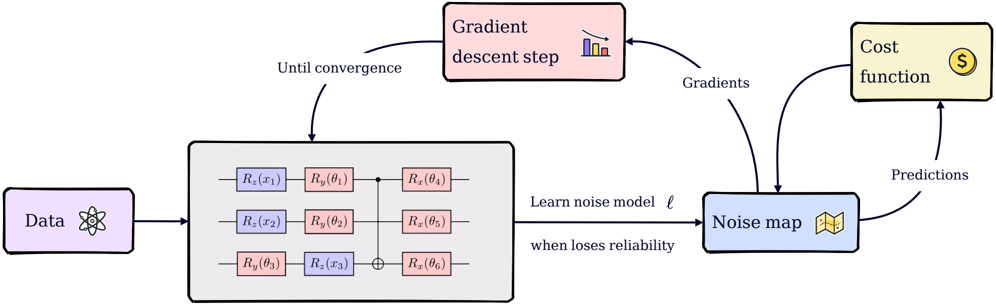


Benchmark library: `https://github.com/qiboteam/qibojit-benchmarks`

see Pasquale's talk

A full-stack example

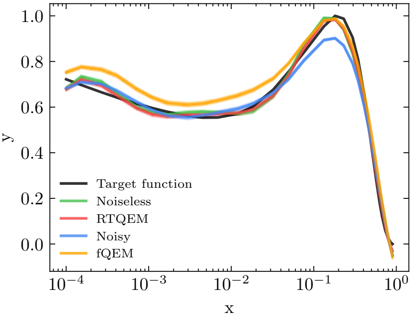
We define a Real-Time Quantum Error Mitigation (RTQEM) procedure.

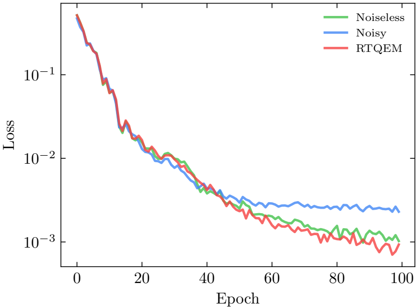


- consider a Variational Quantum Algorithm trained with gradient descent;
- learn the noise map ℓ every time is needed over the procedure;
- use ℓ to clean up both predictions and gradients.

One dimensional HEP target: the u-quark PDF 📖 arXiv:2311.05680

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





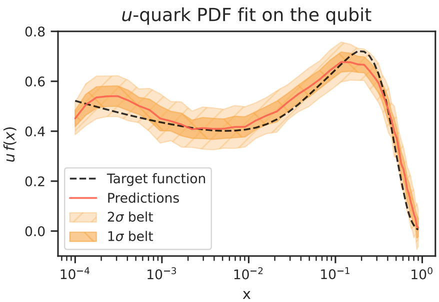
1. thanks to the RTQEM procedure, we reach a good minimum of the cost function;
2. the QEM is not effective is applied to a corrupted scenario (orange curve).

1. High-Level API (Qibo)

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

2. Execution on hardware

- 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	$< 2\text{h}$

see Robbiati's talk

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.