Non-Conventional Computing for HEP

Challenges and opportunities from novel computing technologies

Stefano Carrazza

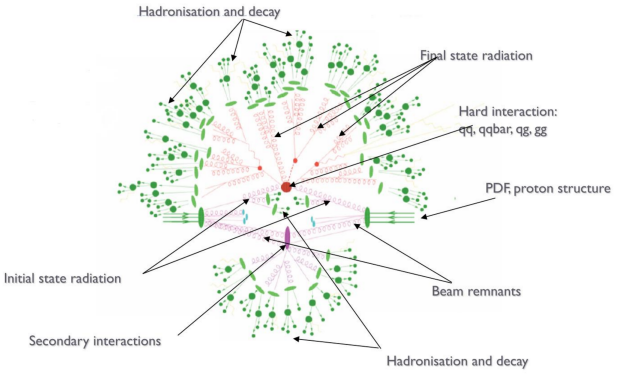
New Frontiers in Theoretical Physics - XXXVIII Convegno Nazionale di Fisica Teorica

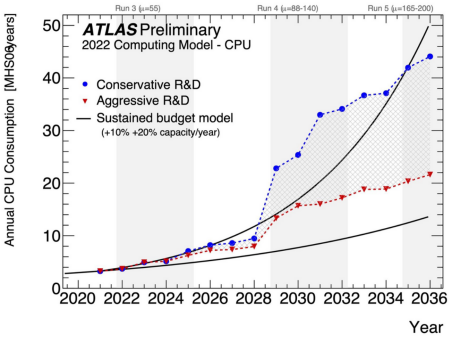
Cortona, May 22th, 2025

UNIVERSITÀ DEGLI STUDI DI MILANO

Physics context: Hadronic collisions at the LHC

Monte Carlo event simulation is **computationally very intensive**.





Furthermore, HEP experimentation is fundamentally **stochastic**, therefore an **increase in data collection** will impact the MC production and therefore the computing consumption.

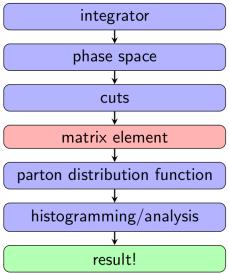
Parton-level Monte Carlo generators

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:

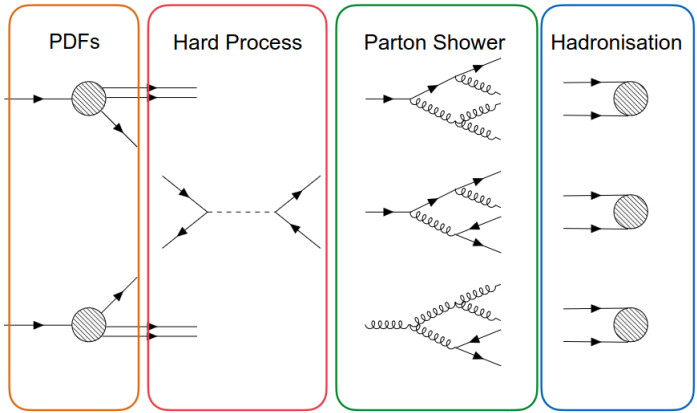
- $|\mathcal{M}|$ is the matrix element,
- $f_i(x, Q^2)$ are Parton Distribution Functions (PDFs),
- $\{p_n\}$ phase space for n particles,
- $\mathcal{J}_m^n$ jet function for n particles to m.



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

Monte Carlo generator pipeline

Schematically, a MC generator can be divided into **modules**:



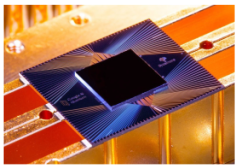
⇒ **Final goal: efficient runtime and low memory usage.**

What about hardware?

Defining hardware acceleration

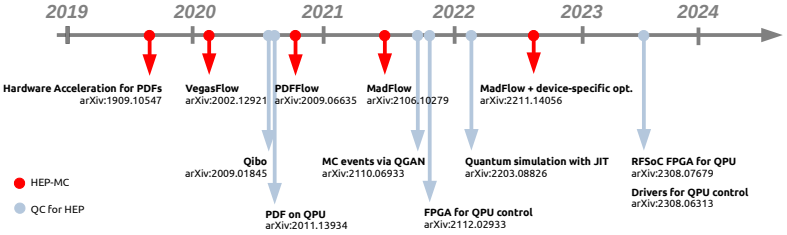
R&D software for new technologies, such as **hardware accelerators**:

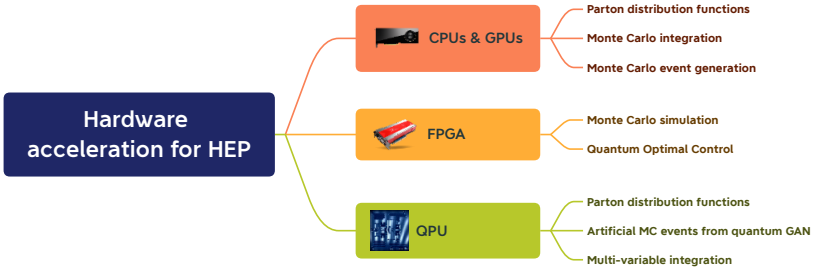
CPU **GPU** **FPGA/ASIC** **Quantum chip**



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

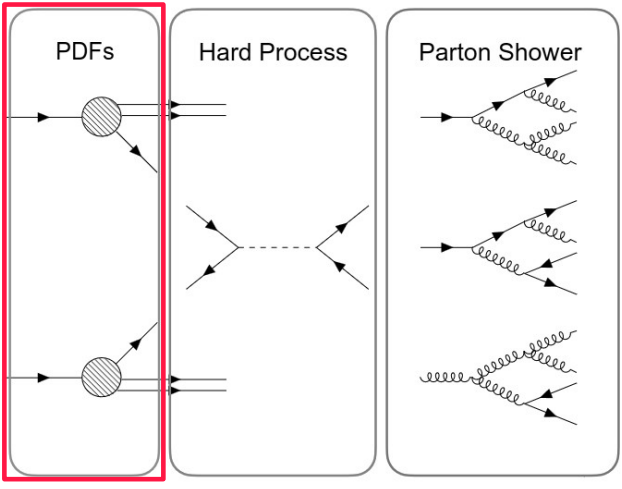
Hardware accelerators in HEP





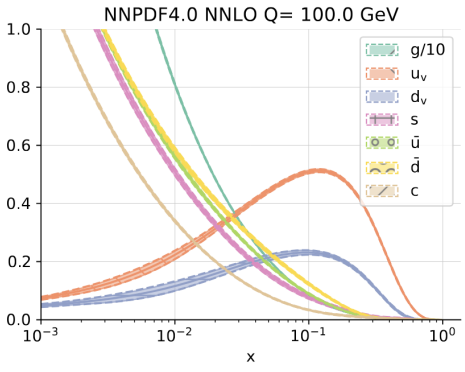
MC event generation on GPU

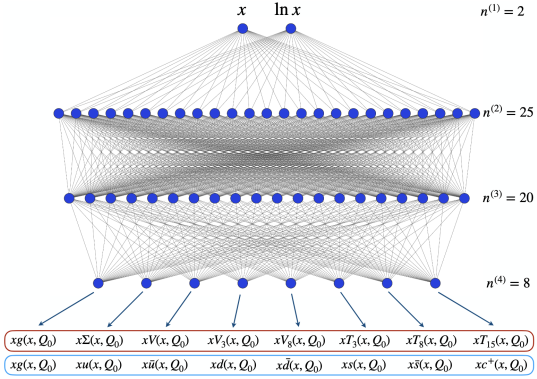
Example: Parton Distribution Functions



Parton distribution functions (Machine Learning)

An unbiased determination of PDFs and its uncertainties are crucial for predictions.





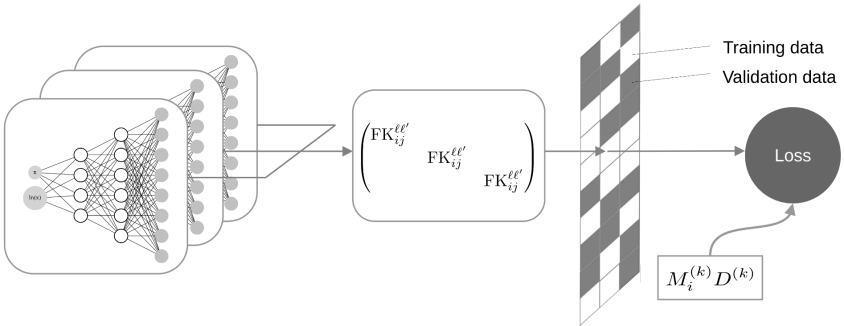
The NNPDF approach uses neural networks and a full machine learning training framework:

$$xf_k(x)(x,Q_0^2;\boldsymbol{\theta}) = A_k x^{1-\alpha_k}(1-x)^{\beta_k}\mathrm{NN}_k(x;\boldsymbol{\theta}), \quad k = \{g,u,\bar{u},d,\bar{d},s,\bar{s},c^+\}.$$

PDF determination requires:

- Uncertainty estimation through multiple fits of Monte Carlo replicas.
- Prediction evaluation through convolutions with matrix elements and evolution operators.

$$\sigma_{pp\to X} = \sum_{k,l} \sum_{\delta,\gamma} \tilde{W}_{kl,\delta,\gamma} f_k(x_\delta, Q_0^2) f_l(x_\gamma, Q_0^2)$$

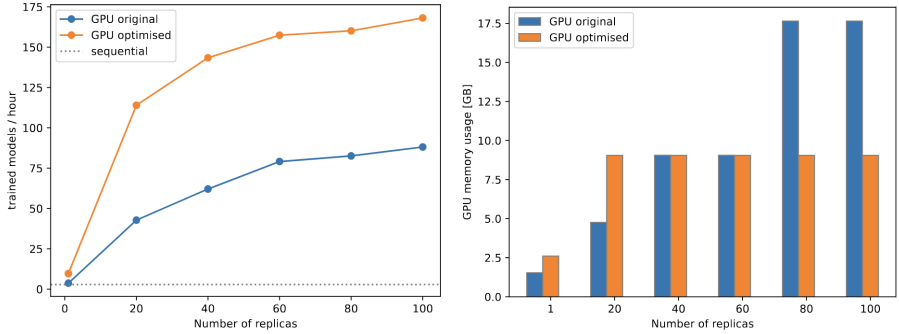


Both operations are parallelized on GPU.

Boosting NNPDF fits on GPU - Results

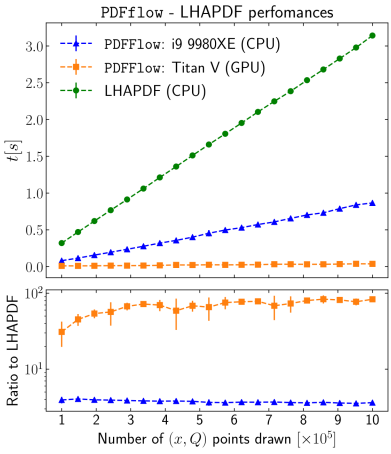
arXiv:2410.16248

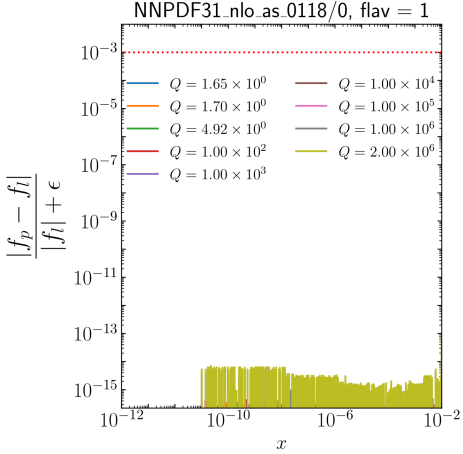
Significant reduction in PDF training time on GPUs (e.g., NVIDIA H100):



Up to 80% runtime improvement when compared to CPU performance (sequential):
Increase of models per hours $\rightarrow$ better hyper parameter fine tuning $\rightarrow$ better PDFs

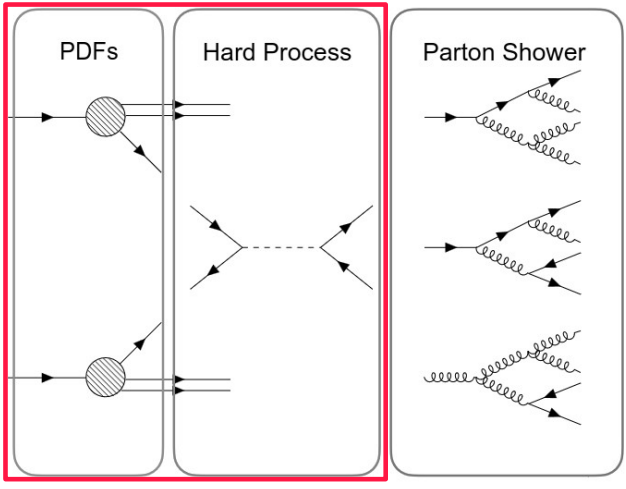
PDF interpolation and inference on GPU is required for MC simulation:





The LHAPDF implementation on GPU (`PDFFlow`) enables this possibility.

Example 2: Event generation

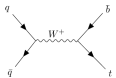


Event generation

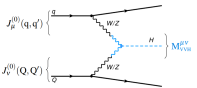
GPU-aware integration wrapper: VegasFlow

arXiv:2010.09341

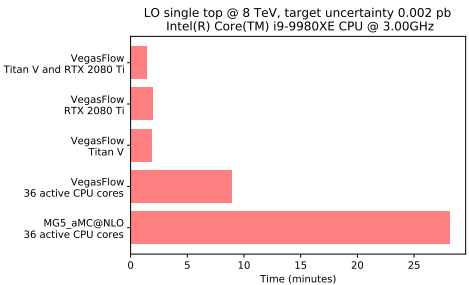
Combine `PDFFlow` and `VegasFlow` (MC integrator, 10.1016/j.cpc.2020.107376)



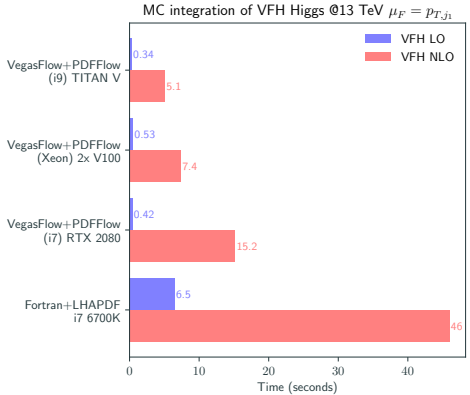
Born diagram for $q\bar{q} \to \bar{b}t$



Born diagram for $qQ \to qQH$



- ✓ Best speed-up at LO: 19x
- ✓ Best speed-up at NLO: 9x

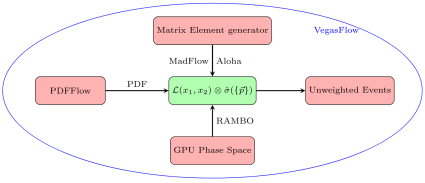


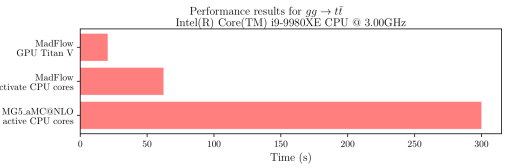
Time per iteration
(C) consumer-grade
(P) professional-grade hardware
CPU implementation: `LHAPDF` + Fortran code
GPU implementation: `PDFFlow` + `VegasFlow`

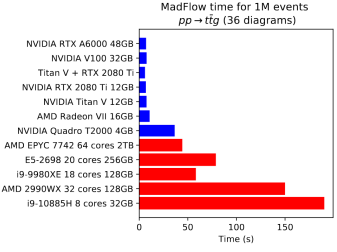
Beyond process-dependent code: MadFlow

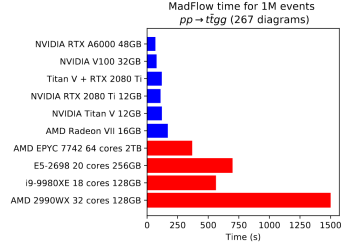
arXiv:2106.10279

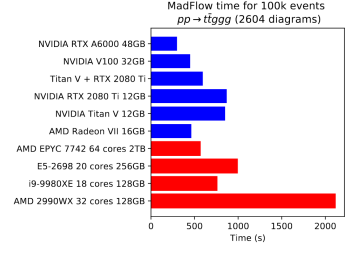
Extend MadGraph interface to write the program combining `PDFFlow+VegasFlow` and `MadGraph`.







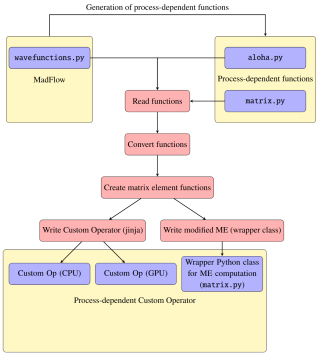


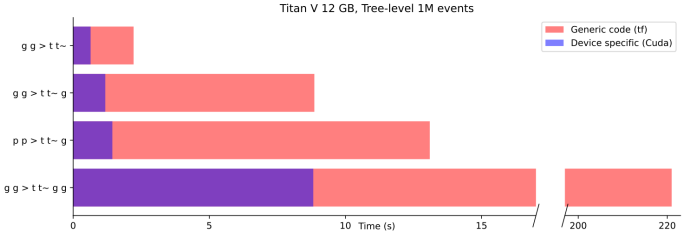


Beyond hardware agnostic code: overoptimization

arXiv:2211.14056

Easily extensible and interfaceable with other languages (C++, Cuda, Fortran, Rust).

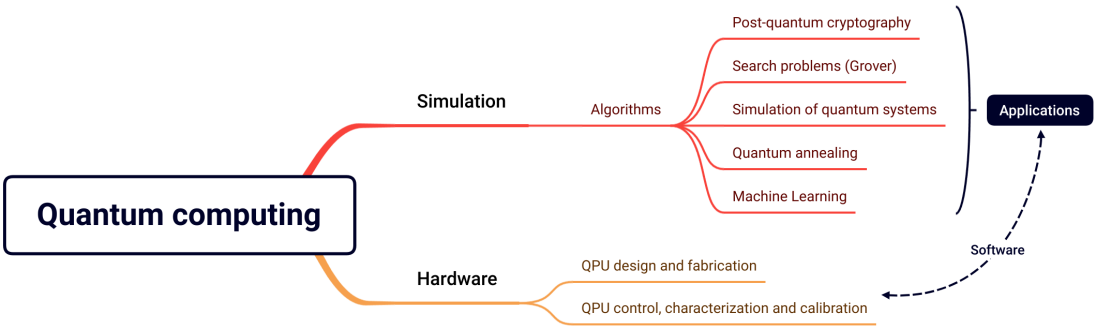




Since in this case the bottleneck is created by the sheer amount of diagrams, we can write a transpiler so that we can convert them in CUDA code that gets compiled before running the process.

What about quantum hardware?

Software and Quantum Computing



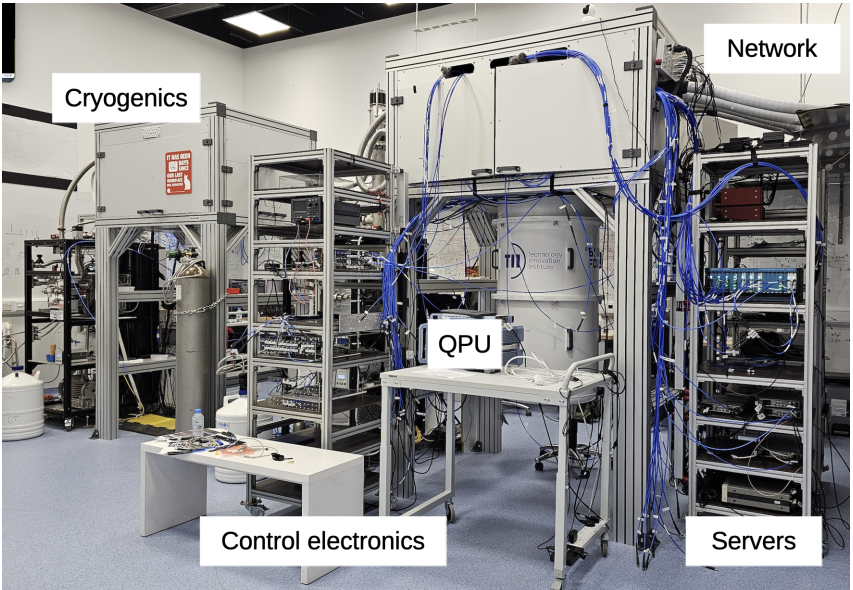
Simulation

- required to develop algorithms
- complete introspection
- require noise modeling

Hardware

- limited (in many senses)
- requires calibration
- final validation

The real-world...



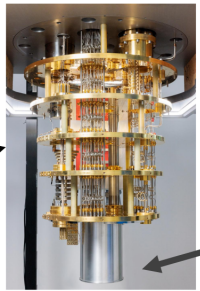
Qubits and Quantum Circuits

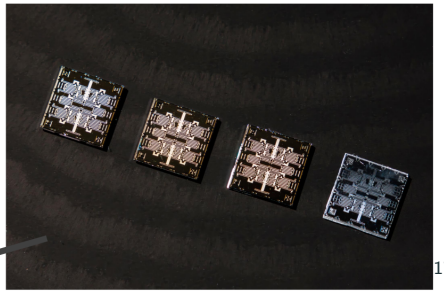
The quantum circuit model considers a sequence of unitary quantum gates:

$$|\psi'\rangle = U_2 U_1 |\psi\rangle \quad \rightarrow \quad |\psi\rangle \text{—} \boxed{U_1} \text{—} \boxed{U_2} \text{—} |\psi'\rangle$$

The final state $|\psi'\rangle$ is given by:

$$\psi'(\boldsymbol{\sigma}) = \sum_{\boldsymbol{\sigma'}} U_1 U_2(\boldsymbol{\sigma}, \boldsymbol{\sigma'}) \psi(\sigma_1, \ldots \sigma'_{i_1}, \ldots, \sigma'_{i_{N_{\text{targets}}}}, \ldots, \sigma_N),$$





Quantum gates

- **Single-qubit gates**
  - Pauli gates
  - Hadamard gate
  - Phase shift gate
  - Rotation gates
- **Two-qubit gates**
  - Controlled gates
  - Swap gate
  - fSim gate
- **Three-qubit gates**
  - Toffoli

Operator	Gate(s)	Matrix
Pauli-X (X)		$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y (Y)		$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli-Z (Z)		$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard (H)		$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
Phase (S, P)		$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$
$\pi/8$ (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)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$
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)		$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$

Pauli gates

X gate

The X gate acts like the classical NOT gate, it is represented by the σ_x matrix,

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

therefore

$$|0\rangle \text{ —} \boxed{X} \text{— } |1\rangle$$

$$|1\rangle \text{ —} \boxed{X} \text{— } |0\rangle$$

Z gate

The Z gate flips the sign of $|1\rangle$, it is represented by the σ_z matrix,

$$\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

therefore

$$|0\rangle \text{ —} \boxed{Z} \text{— } |0\rangle$$

$$|1\rangle \text{ —} \boxed{Z} \text{— } -|1\rangle$$

Hadamard gate

The Hadamard gate (H gate) is defined as

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

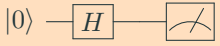
Therefore it creates a superposition of states

$$|0\rangle \text{ —}\boxed{H}\text{— } \frac{|0\rangle + |1\rangle}{\sqrt{2}} \equiv |+\rangle$$

$$|1\rangle \text{ —}\boxed{H}\text{— } \frac{|0\rangle - |1\rangle}{\sqrt{2}} \equiv |-\rangle$$

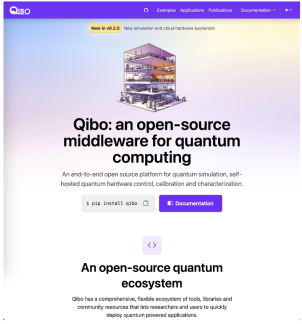
Measurement (M) gate:

Lets consider the following circuit:



When measuring the final state we obtain 0 or 1 each with 50% probability.

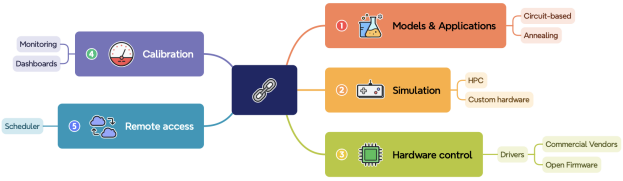
In 2020 we introduced Qibo as a modular and open-source framework for quantum computing, control and calibration.



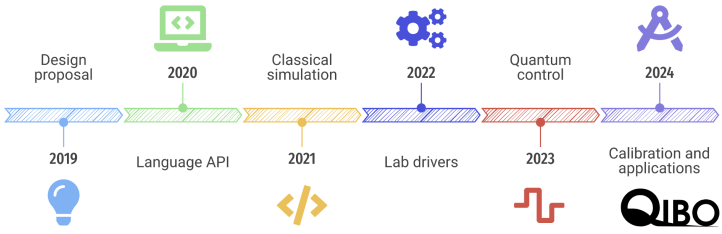
https://qibo.science

Qibo collaboration, arXiv:2009.01845

Middleware challenges

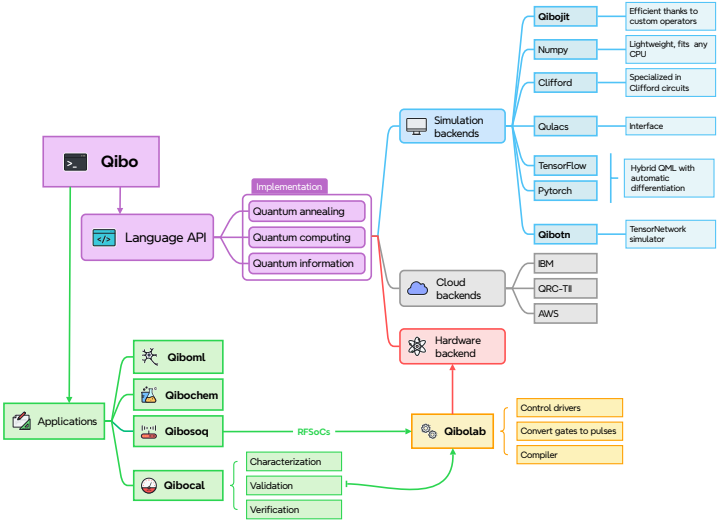


Qibo development timeline

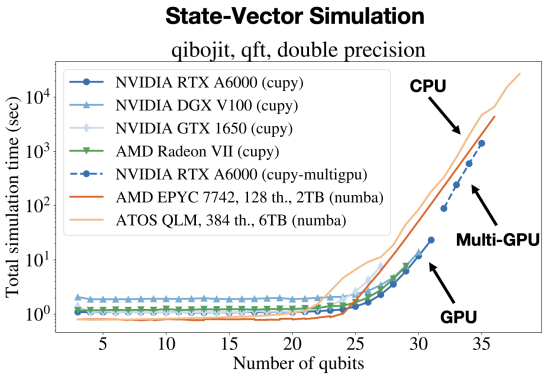


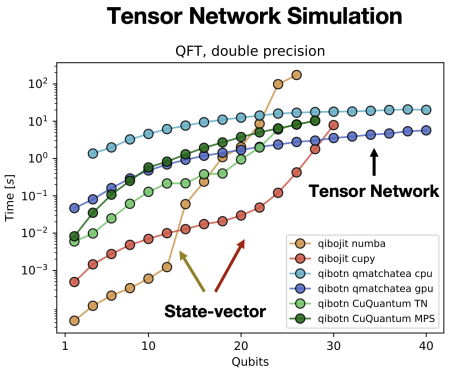
Quantum Middleware

arXiv:2009.01845



Classical quantum simulation is an important tool for algorithm design:





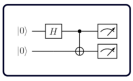
Simulation strategies are limited by memory and performance.

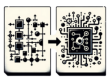
Quantum Control and Calibration

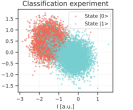
arXiv:2308.06313

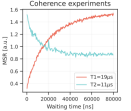
In 2023 we introduced Qibolab as a modular and open-source library for quantum control.

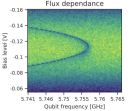
In 2024 we presented Qibocal, a software tool for quantum characterization and calibration of qubits.

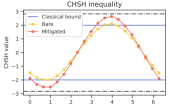


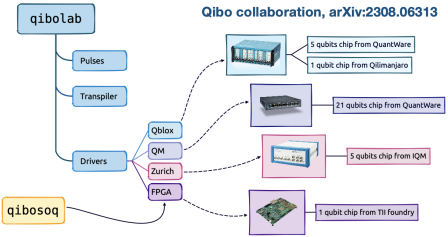


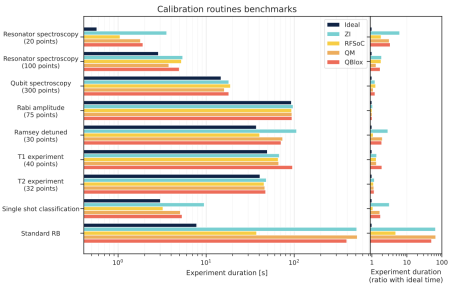










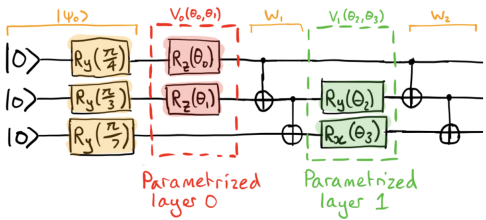


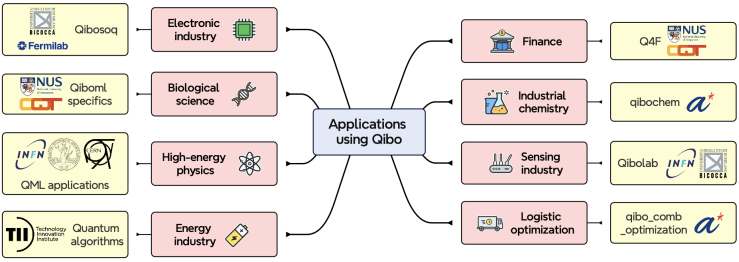
Algorithms for QPUs

Applications

Qibo provide standardized tools to design quantum applications based on quantum digital and analog computing, including:

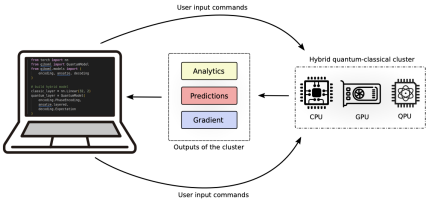
- Variational Quantum Circuits (VQC)
- Variational Quantum Eigensolvers (VQE)
- Quantum machine learning (QML)
- Quantum approximate optimization algorithm (QAOA)
- Quantum error mitigation (QEM)

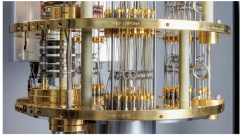




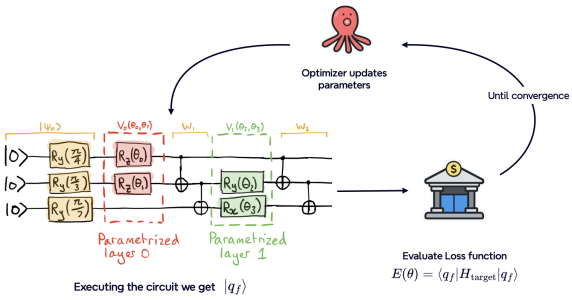
Hybrid quantum-classical algorithms

Quantum Machine Learning involves quantum process units (QPU) into machine learning (ML) pipelines.





QML Pipeline



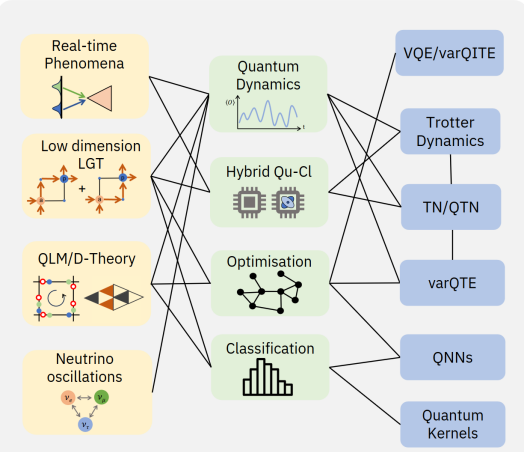
To fully realize the potential of a hybrid quantum-classical cluster, both classical and quantum components must be executable on CPUs, GPUs, and QPUs.

Goal:

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

How?

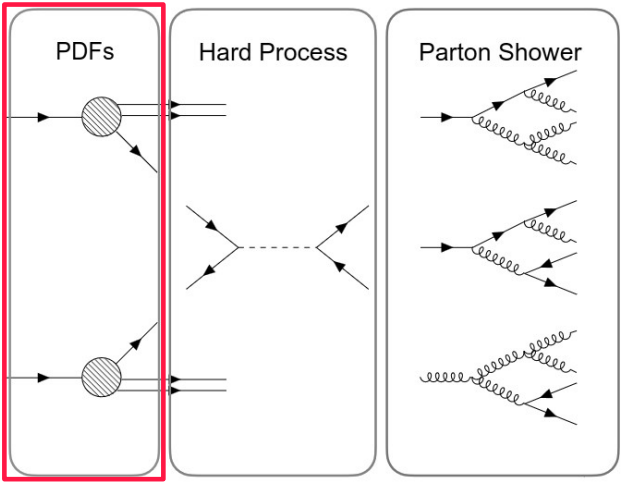
- Designing **hybrid quantum-classical** methods using classical quantum simulation.
- Deploying **classical quantum simulation** techniques on HPC infrastructure.



QC4HEP WG

MC simulation on QPU

Example: Parton Distribution Functions



Parton distribution functions (Machine Learning)

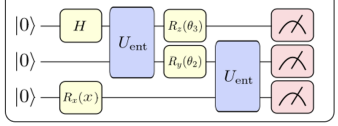
Quantum Regression for PDFs

arXiv:2011.13934

M. Robbiati, A. Sopena, A. Papaluca, SC, arXiv:2311.05680

Quantum Model

A parametric model $U(x;\theta)$



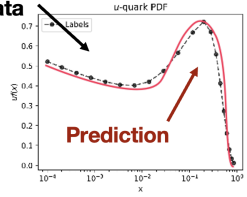
returning predictions

$$\tilde{f}(x;\theta) = \langle 0|U(x;\theta)^\dagger \hat{O} U(x;\theta)|0\rangle$$

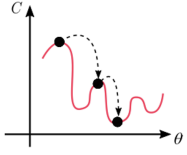
Evaluated on QPU

Classical Data

A target $f(x)$



Loss function (MSE)



A training algorithm

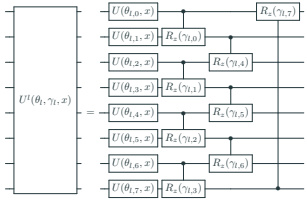
Classical Optimizer (i.e. Gradient Descent)

arXiv:2011.13934

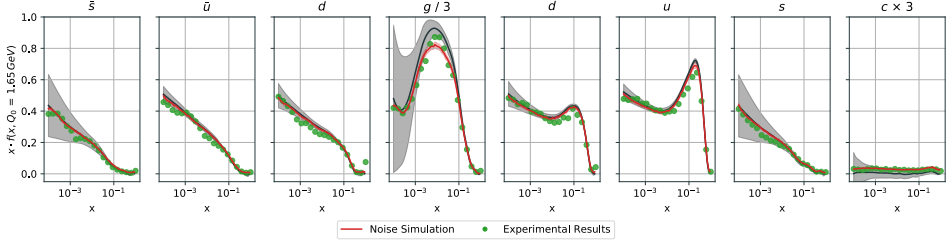
We parametrize **Parton Distribution Functions** with multi-qubit variational quantum circuits:

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**:



Quantum PDFs regression on QPU

arXiv:2011.13934

High level API: Qibo

- </> define **prototypes** and models;
- </> **simulate** training and noise.

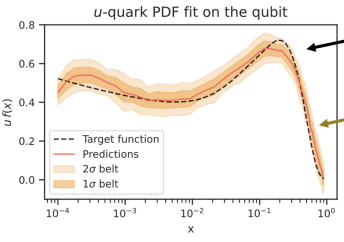
Calibration: Qibocal

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

Execution: Qibolab

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

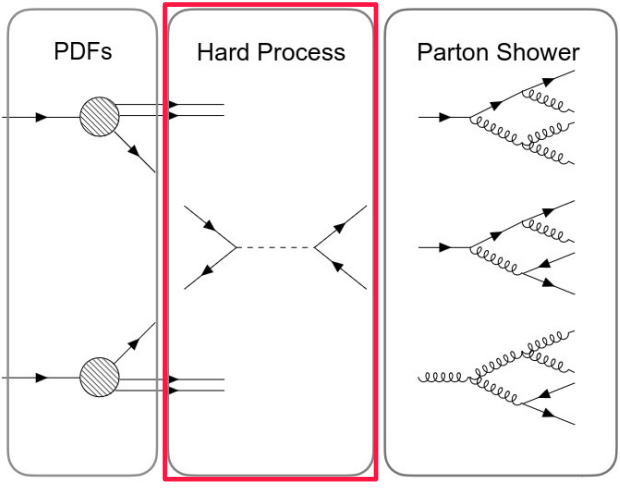
M. Robbiati, A. Sopena, A. Papaluca, SC, arXiv:2311.05680



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

Multi-Control Electronics Support

Example: Monte Carlo Integration / Sampling

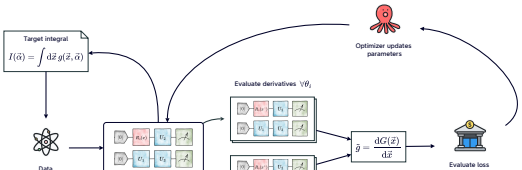


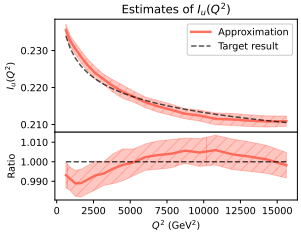
Monte Carlo Integration

Monte Carlo simulation on QPU

arXiv:2308.05657, 2110.06933

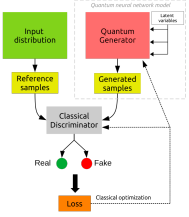
QPUs can be used to parametrize integrands and integrals via:

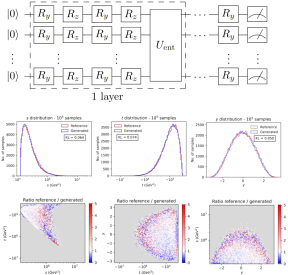




Quantum GANs:

QPUs can sample Monte Carlo events using quantum generative adversarial architectures:





Outlook

Conclusion: Embracing Non-Conventional Computing

- HEP's intense computing needs drive exploration of novel technologies.
- **GPUs** are delivering significant speedups in MC generation and analysis (`PDFFlow` , `VegasFlow` , `MadFlow`).
- Hardware-aware software optimization maximizes GPU potential.
- **Quantum Computing** offers future possibilities for theoretical modeling and data analysis.
- Frameworks like **Qibo** facilitate quantum algorithm development for HEP.
- Hybrid quantum-classical approaches are key for near-term quantum applications.
- Continued progress in non-conventional computing is vital for HEP's future.

Thank you!