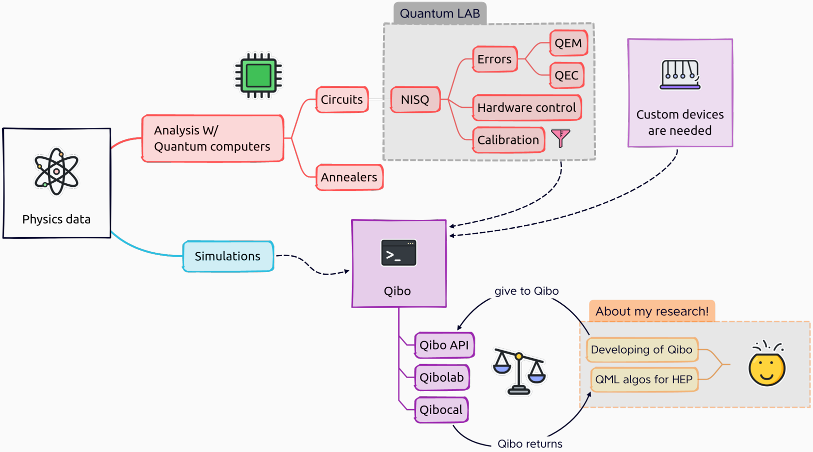
Towards a full-stack quantum operating system

Quantum simulation, control and calibration using `qibo`

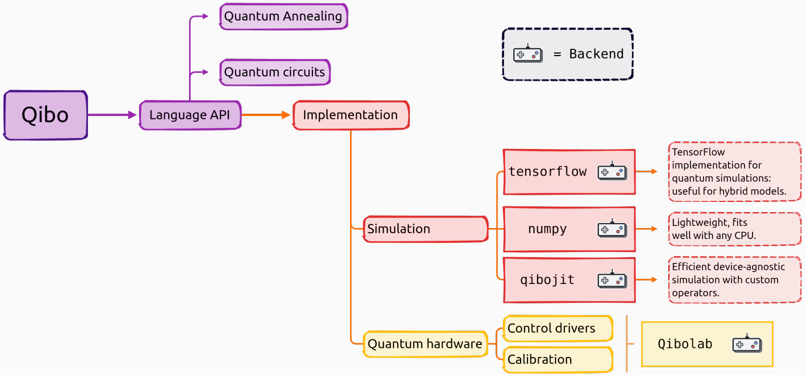
Matteo Robbiati

10 May 2023

Working in the NISQ era



What is qibo?



arXiv:2009.01845: *"Qibo: a framework for quantum simulation with hardware acceleration."*

Some features

More about `qibojit`

- We do state vector simulation, which solves:

$$\psi'(\sigma_1, ..., \sigma_n) = \sum_{\tau'} G(\tau, \tau')\psi(\sigma_1, ..., \tau', ..., \sigma_n), \quad (1)$$

More about `qibojit`

- We do state vector simulation, which solves:

$$\psi'(\sigma_1, ..., \sigma_n) = \sum_{\tau'} G(\tau, \tau')\psi(\sigma_1, ..., \tau', ..., \sigma_n), \tag{1}$$

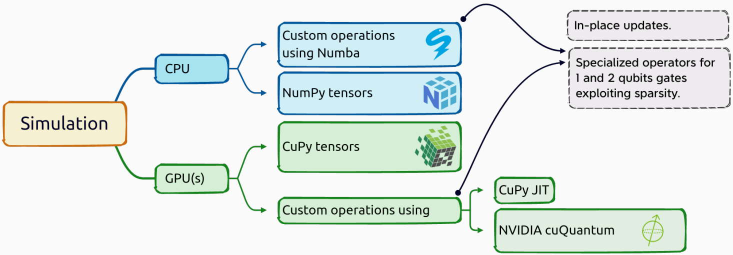
- where the number of operations scales exponentially with N_{qubits}.

More about `qibojit`

- We do state vector simulation, which solves:

$$\psi'(\sigma_1, ..., \sigma_n) = \sum_{\tau'} G(\tau, \tau')\psi(\sigma_1, ..., \tau', ..., \sigma_n), \tag{1}$$

- where the number of operations scales exponentially with N_{qubits}.
- For this reason we built `qibojit` (recommended if $N_{qubits \geq 20}$):



arXiv:2203.08826: *"Quantum simulation with just-in-time compilation."*

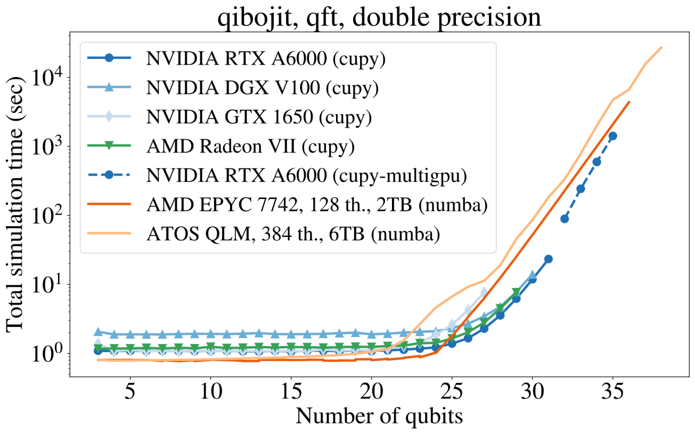


Figure 1: Quantum Fourier Transform execution with `qibojit` backend for growing number of qubits.

More about quantum annealing with `qibo`

- An `AdiabaticEvolution` model is provided, which implements:

$$H_{\mathrm{ad}}(\tau; \boldsymbol{\theta}) = \left[1 - s(\tau; \boldsymbol{\theta})\right] H_0 + s(\tau; \boldsymbol{\theta}) H_1. \tag{2}$$

More about quantum annealing with `qibo`

- An `AdiabaticEvolution` model is provided, which implements:

$$H_{\mathrm{ad}}(\tau;\boldsymbol{\theta}) = \left[1 - s(\tau;\boldsymbol{\theta})\right] H_0 + s(\tau;\boldsymbol{\theta}) H_1. \tag{2}$$

- which can be used by:

More about quantum annealing with `qibo`

- An `AdiabaticEvolution` model is provided, which implements:

$$H_{\mathrm{ad}}(\tau;\boldsymbol{\theta}) = \left[1 - s(\tau;\boldsymbol{\theta})\right] H_0 + s(\tau;\boldsymbol{\theta}) H_1. \tag{2}$$

- which can be used by:
  - defining H_0 and H_1 symbolically (we use sympy);

More about quantum annealing with `qibo`

- An `AdiabaticEvolution` model is provided, which implements:

$$H_{\mathrm{ad}}(\tau;\boldsymbol{\theta}) = \left[1 - s(\tau;\boldsymbol{\theta})\right] H_0 + s(\tau;\boldsymbol{\theta}) H_1. \tag{2}$$

- which can be used by:
  - defining H_0 and H_1 symbolically (we use `sympy`);
  - defining a scheduling function $s(\tau;\boldsymbol{\theta})$ and a timestep `dt`;

More about quantum annealing with qibo

- An `AdiabaticEvolution` model is provided, which implements:

$$H_{\rm ad}(\tau;\boldsymbol{\theta}) = \left[1 - s(\tau;\boldsymbol{\theta})\right] H_0 + s(\tau;\boldsymbol{\theta}) H_1. \tag{2}$$

- which can be used by:
  - defining H_0 and H_1 symbolically (we use `sympy`);
  - defining a scheduling function $s(\tau;\boldsymbol{\theta})$ and a timestep `dt`;
  - setting the `solver` to use for integrating Schrondiger's equation.

More about quantum annealing with qibo

- An `AdiabaticEvolution` model is provided, which implements:

$$H_{\mathrm{ad}}(\tau;\boldsymbol{\theta}) = \left[1 - s(\tau;\boldsymbol{\theta})\right] H_0 + s(\tau;\boldsymbol{\theta}) H_1. \tag{2}$$

- which can be used by:
  - defining H_0 and H_1 symbolically (we use `sympy`);
  - defining a scheduling function $s(\tau;\boldsymbol{\theta})$ and a timestep `dt`;
  - setting the `solver` to use for integrating Schrondiger's equation.
  - calling the `AdiabaticEvolution` object at some final time T.

More about quantum annealing with `qibo`

- An `AdiabaticEvolution` model is provided, which implements:

$$H_{\mathrm{ad}}(\tau;\boldsymbol{\theta}) = \left[1 - s(\tau;\boldsymbol{\theta})\right] H_0 + s(\tau;\boldsymbol{\theta}) H_1. \tag{2}$$

- which can be used by:
  - defining H_0 and H_1 symbolically (we use sympy);
  - defining a scheduling function $s(\tau;\boldsymbol{\theta})$ and a timestep `dt`;
  - setting the `solver` to use for integrating Schrondiger's equation.
  - calling the `AdiabaticEvolution` object at some final time T.

- This mechanism "pushes" the state during the evolution by sequentially executing a circuit obtained by trotterizing H_{ad}.

More about quantum annealing with `qibo`

- An `AdiabaticEvolution` model is provided, which implements:

$$H_{\mathrm{ad}}(\tau;\boldsymbol{\theta}) = \left[1 - s(\tau;\boldsymbol{\theta})\right] H_0 + s(\tau;\boldsymbol{\theta}) H_1. \tag{2}$$

- which can be used by:
  - defining H_0 and H_1 symbolically (we use `sympy`);
  - defining a scheduling function $s(\tau;\boldsymbol{\theta})$ and a timestep `dt`;
  - setting the `solver` to use for integrating Schrondiger's equation.
  - calling the `AdiabaticEvolution` object at some final time T.

- This mechanism "pushes" the state during the evolution by sequentially executing a circuit obtained by trotterizing H_{ad}.

- If `solver=="exp"`, we use the evolutionary operator[1]:

$$|\psi(\tau = j\mathrm{d}t)\rangle = \overleftarrow{\prod_j} U_j \,|\psi(\tau = 0)\rangle \tag{3}$$

[1] Translated into a circuit form using the Trotter decomposition.

Adiabatic evolution on qibo backends

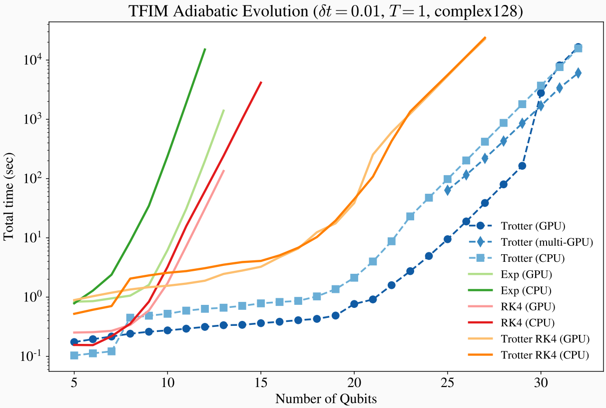
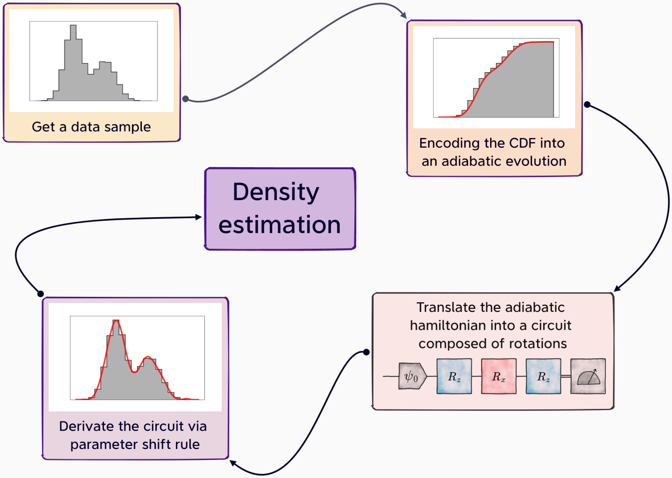


Figure 2: Adiabatic evolution execution with growing number of qubits and different solvers.

A full-stack QML algorithm

The theoretical idea



SIMULATION: fit CDF with Adiabatic Evolution (AE)

- Given a sample $\{x\}$ and calculated its CDF values $F(x)$:

SIMULATION: fit CDF with Adiabatic Evolution (AE)

- Given a sample $\{x\}$ and calculated its CDF values $F(x)$:
  - we select two hamiltonians H_0 and H_1 such that a target observable has energy $E = 0$ and $E = 1$ respectively on H_0 and H_1 ground states;

SIMULATION: fit CDF with Adiabatic Evolution (AE)

- Given a sample $\{x\}$ and calculated its CDF values $F(x)$:
  - we select two hamiltonians H_0 and H_1 such that a target observable has energy $E = 0$ and $E = 1$ respectively on H_0 and H_1 ground states;
  - we map $(x, F) \rightarrow (\tau, E)$.

SIMULATION: fit CDF with Adiabatic Evolution (AE)

- Given a sample $\{x\}$ and calculated its CDF values $F(x)$:
  - we select two hamiltonians H_0 and H_1 such that a target observable has energy $E = 0$ and $E = 1$ respectively on H_0 and H_1 ground states;
  - we map $(x, F) \to (\tau, E)$.
- The AE training strategy follows:

SIMULATION: fit CDF with Adiabatic Evolution (AE)

- Given a sample $\{x\}$ and calculated its CDF values $F(x)$:
  - we select two hamiltonians H_0 and H_1 such that a target observable has energy $E = 0$ and $E = 1$ respectively on H_0 and H_1 ground states;
  - we map $(x, F) \rightarrow (\tau, E)$.
- The AE training strategy follows:
  1. we run the evolution with random initial $\boldsymbol{\theta}_0$ into the scheduling;

SIMULATION: fit CDF with Adiabatic Evolution (AE)

- Given a sample $\{x\}$ and calculated its CDF values $F(x)$:
  - we select two hamiltonians H_0 and H_1 such that a target observable has energy $E = 0$ and $E = 1$ respectively on H_0 and H_1 ground states;
  - we map $(x, F) \to (\tau, E)$.
- The AE training strategy follows:
  1. we run the evolution with random initial $\boldsymbol{\theta}_0$ into the scheduling;
  2. we track the energy of a Pauli Z during the evolution;

SIMULATION: fit CDF with Adiabatic Evolution (AE)

- Given a sample $\{x\}$ and calculated its CDF values $F(x)$:
  - we select two hamiltonians H_0 and H_1 such that a target observable has energy $E = 0$ and $E = 1$ respectively on H_0 and H_1 ground states;
  - we map $(x, F) \rightarrow (\tau, E)$.
- The AE training strategy follows:
  1. we run the evolution with random initial $\boldsymbol{\theta}_0$ into the scheduling;
  2. we track the energy of a Pauli Z during the evolution;
  3. we calculate a loss function $J_{\rm mse}$:

$$J_{\rm mse} = \sum_{k=1}^{N_{\rm sample}} \big[E(\tau_k) - F(x_k)\big]^2;$$

SIMULATION: fit CDF with Adiabatic Evolution (AE)

- Given a sample $\{x\}$ and calculated its CDF values $F(x)$:
  - we select two hamiltonians H_0 and H_1 such that a target observable has energy $E = 0$ and $E = 1$ respectively on H_0 and H_1 ground states;
  - we map $(x, F) \to (\tau, E)$.
- The AE training strategy follows:
  1. we run the evolution with random initial $\boldsymbol{\theta}_0$ into the scheduling;
  2. we track the energy of a Pauli Z during the evolution;
  3. we calculate a loss function $J_{\rm mse}$:

$$J_{\rm mse} = \sum_{k=1}^{N_{\rm sample}} \left[E(\tau_k) - F(x_k)\right]^2;$$

  4. we choose an optimizer to find $\boldsymbol{\theta}_{\rm best}$ which minimizes $J_{\rm mse}$.

SIMULATION: fit CDF with Adiabatic Evolution (AE)

- Given a sample $\{x\}$ and calculated its CDF values $F(x)$:
  - we select two hamiltonians H_0 and H_1 such that a target observable has energy $E = 0$ and $E = 1$ respectively on H_0 and H_1 ground states;
  - we map $(x, F) \to (\tau, E)$.
- The AE training strategy follows:
  1. we run the evolution with random initial $\boldsymbol{\theta}_0$ into the scheduling;
  2. we track the energy of a Pauli Z during the evolution;
  3. we calculate a loss function J_{mse}:

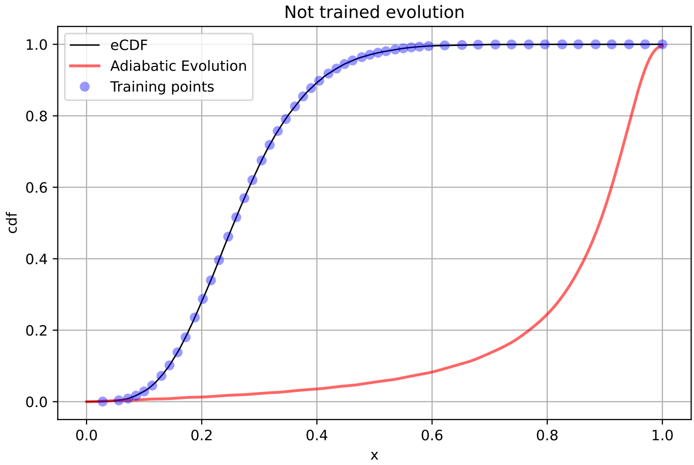
$$J_{\mathrm{mse}} = \sum_{k=1}^{N_{\mathrm{sample}}} \left[E(\tau_k) - F(x_k)\right]^2;$$

  4. we choose an optimizer to find $\boldsymbol{\theta}_{\mathrm{best}}$ which minimizes J_{mse}.

arXiv:2303.11346: *"Determining probability density functions with adiabatic quantum computing."*

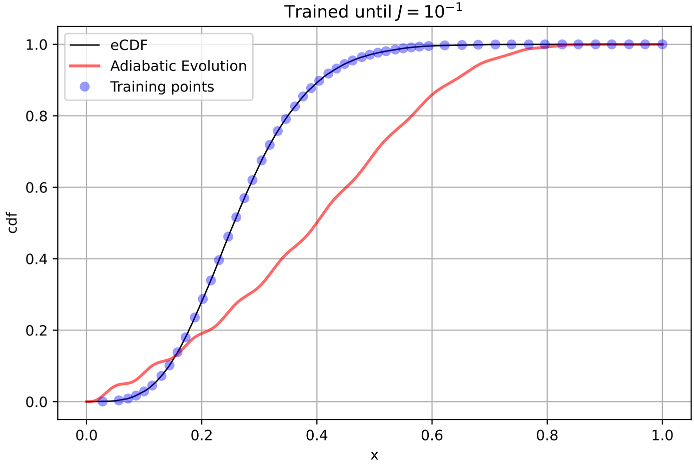
A toy example with `nqubits=1` - starting point

- `nparams=20`, `dt=0.1`, `final_time=50` , `target_loss=None`



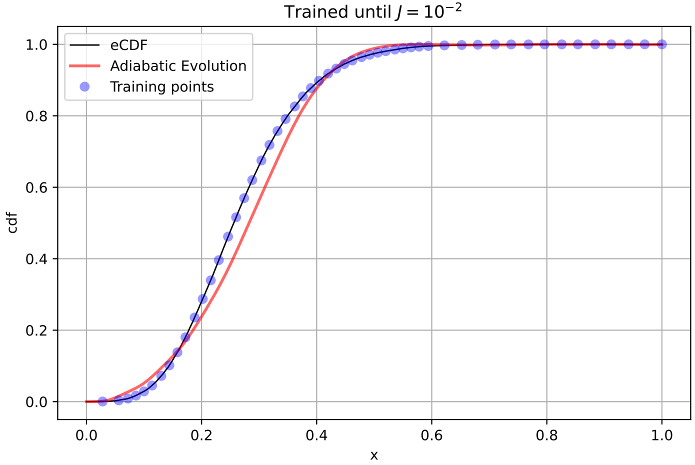
A toy example - until $J_{\mathrm{MSE}} = 10^{-1}$

- `nparams=20`, `dt=0.1`, `final_time=50` , `target_loss=1e-1`



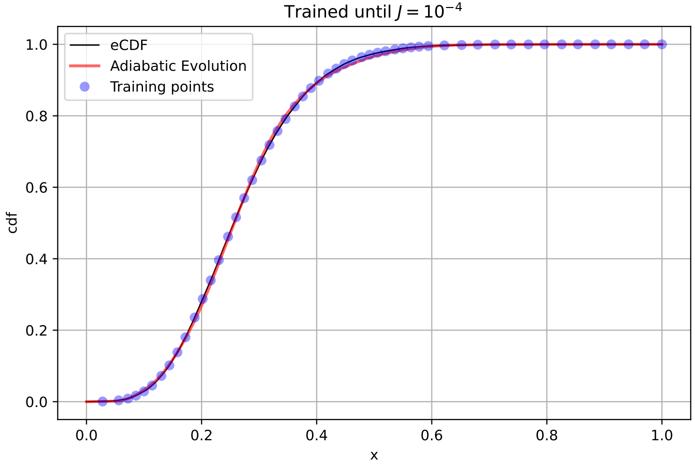
A toy example - until $J_{\mathrm{MSE}} = 10^{-2}$

- `nparams=20`, `dt=0.1`, `final_time=50` , `target_loss=1e-2`



A toy example - ending at $J_{\mathrm{MSE}} = 10^{-4}$

- `nparams=20`, `dt=0.1`, `final_time=50` , `target_loss=1e-4`



SIMULATION: from $\{H_{\mathrm{ad}}\}$ to a circuit and derivate!

SIMULATION: from $\{H_{\text{ad}}\}$ to a circuit and derivate!

- Firstly, we did some calculations and approximations in order to:

SIMULATION: from $\{H_{\mathrm{ad}}\}$ to a circuit and derivate!

- Firstly, we did some calculations and approximations in order to:

1. translate the Hamiltonians' sequence into a single unitary:

$$\prod_{j=1}^{n} e^{-iH_j \mathrm{d}t} \rightarrow \mathcal{U}(t);$$

SIMULATION: from $\{H_{\mathrm{ad}}\}$ to a circuit and derivate!

- Firstly, we did some calculations and approximations in order to:

1. translate the Hamiltonians' sequence into a single unitary:

$$\prod_{j=1}^{n} e^{-iH_j \mathrm{d}t} \rightarrow \mathcal{U}(t);$$

2. translate this unitary in a sequence of rotational gates:

$$\mathcal{U}(t) = R_z(\theta_1)R_x(\theta_2)R_z(\theta_3) \qquad \text{with } \theta_i \equiv \theta_i(t).$$

SIMULATION: from $\{H_{\mathrm{ad}}\}$ to a circuit and derivate!

- Firstly, we did some calculations and approximations in order to:

  1. translate the Hamiltonians' sequence into a single unitary:

  $$\prod_{j=1}^{n} e^{-iH_j \mathrm{d}t} \rightarrow \mathcal{U}(t);$$

  2. translate this unitary in a sequence of rotational gates:

  $$\mathcal{U}(t) = R_z(\theta_1) R_x(\theta_2) R_z(\theta_3) \qquad \text{with } \theta_i \equiv \theta_i(t).$$

- Then, we derivate the expected values using parameter shift rule and chain rule.

SIMULATION: from $\{H_{\mathrm{ad}}\}$ to a circuit and derivate!

- Firstly, we did some calculations and approximations in order to:

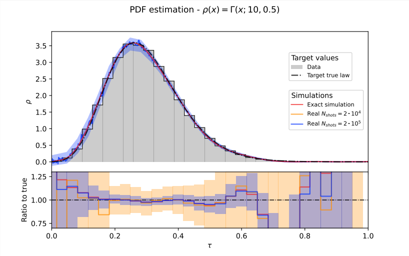
  1. translate the Hamiltonians' sequence into a single unitary:

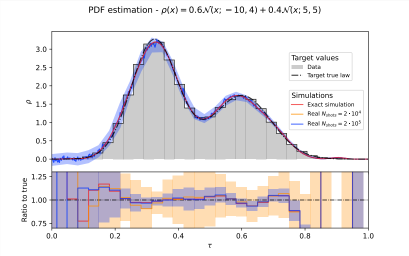
  $$\prod_{j=1}^{n} e^{-iH_j \mathrm{d}t} \rightarrow \mathcal{U}(t);$$

  2. translate this unitary in a sequence of rotational gates:

  $$\mathcal{U}(t) = R_z(\theta_1)R_x(\theta_2)R_z(\theta_3) \qquad \text{with } \theta_i \equiv \theta_i(t).$$

- Then, we derivate the expected values using parameter shift rule and chain rule.





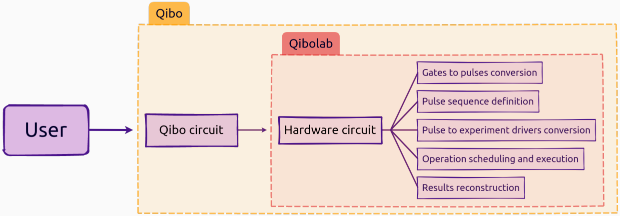
Hardware deployment

EXECUTION: running on the hardware

- qibo is hardware-agnostic!

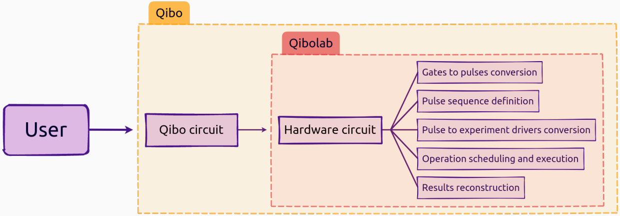
EXECUTION: running on the hardware

- qibo is hardware-agnostic!
- We defined an abstract Platform object, which can be selected via `set_backend("qibolab", platform="my_platform")`.



EXECUTION: running on the hardware

- qibo is hardware-agnostic!
- We defined an abstract Platform object, which can be selected via `set_backend("qibolab", platform="my_platform")`.



- Some labs are already using qibo:

INFN Istituto Nazionale di Fisica Nucleare

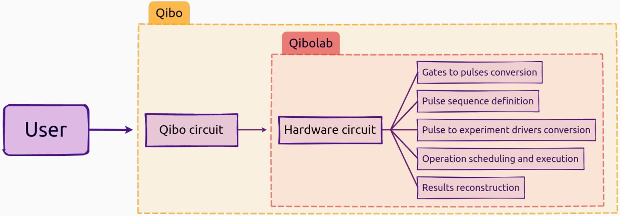
TII Technology Innovation Institute

CQT

BSC

EXECUTION: running on the hardware

- qibo is hardware-agnostic!
- We defined an abstract `Platform` object, which can be selected via `set_backend("qibolab", platform="my_platform")`.



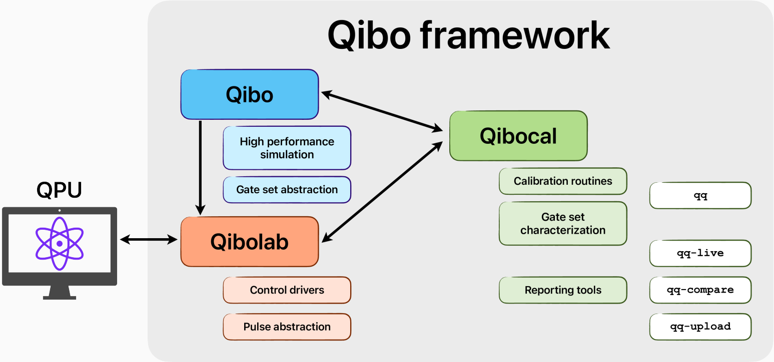
- Some labs are already using qibo:

arXiv:2202.07017: *"An open-source modular framework for quantum computing."*
arXiv:2112.02933: *"ICARUS-Q: Integrated Control and Readout Unit for Scalable Quantum Processors"*

`qibolab` is not enough!

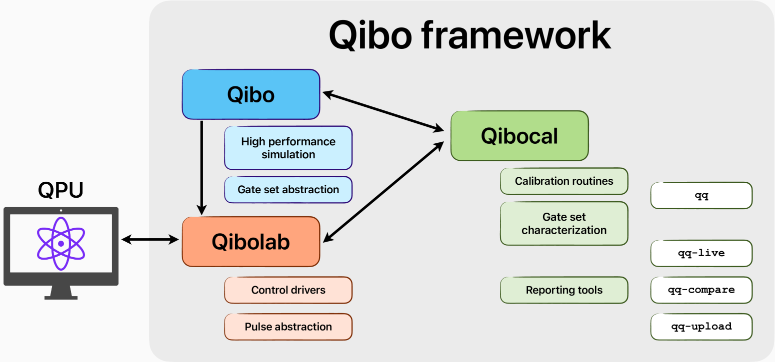
➔ Each quantum control routine is useless if the sequences of pulses are not well calibrated with the single qubit.

- Each quantum control routine is useless if the sequences of pulses are not well calibrated with the single qubit.
- For this reason, `qibocal` was born: a module for quantum calibration and verification.



`qibolab` is not enough!

- Each quantum control routine is useless if the sequences of pulses are not well calibrated with the single qubit.
- For this reason, `qibocal` was born: a module for quantum calibration and verification.



arXiv:2303.10397: *"Towards an open-source framework to perform quantum calibration and characterization."*

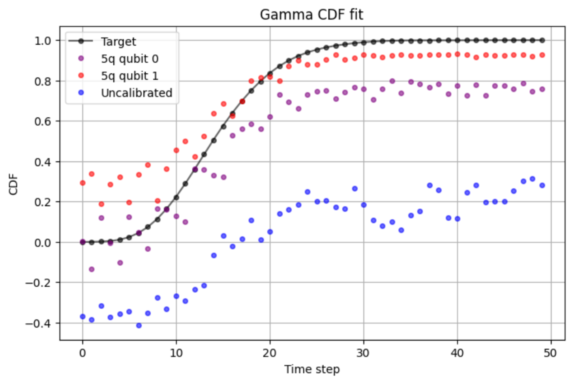


Figure 3: Different qubits requires different calibration and leads to different results.

The importance of `qibocal`

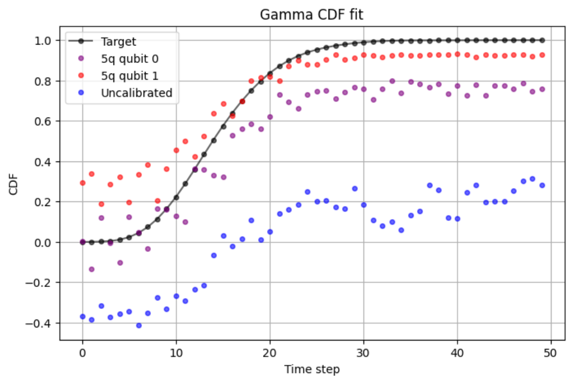


Figure 3: Different qubits requires different calibration and leads to different results.

- Open questions:

The importance of `qibocal`

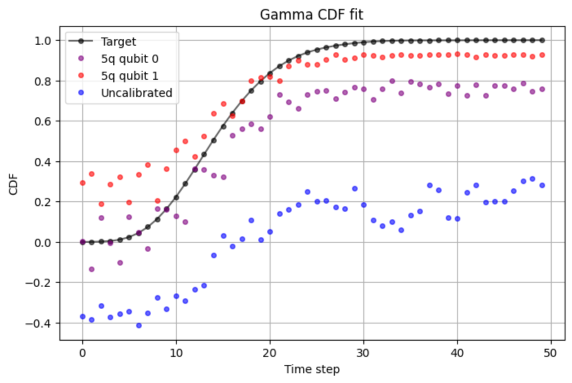


Figure 3: Different qubits requires different calibration and leads to different results.

- Open questions:
  - what if the entire training is performed on a NISQ device? *are the results self-resistent to the noise?*

The importance of `qibocal`

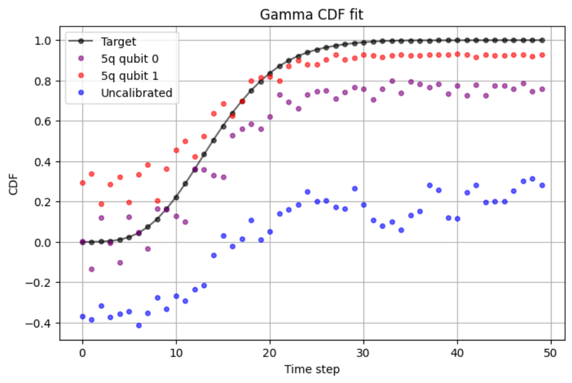


Figure 3: Different qubits requires different calibration and leads to different results.

- Open questions:
  - what if the entire training is performed on a NISQ device? *are the results self-resistent to the noise?*
  - what needed for improving results on the hardware?

☝: what if we train on hardware?

The theoretical idea

➔ Following *Pérez-Salinas et al.* procedure, we can build a *universal quantum regressor* for approximating $y = f(x)$. The model can be:

The theoretical idea

➔ Following *Pérez-Salinas et al.* procedure, we can build a *universal quantum regressor* for approximating $y = f(x)$. The model can be:

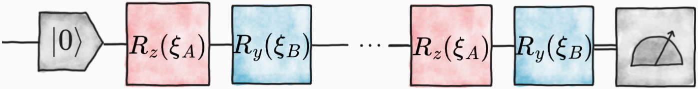


Figure 4: Here $\xi_A = \theta_1 x + \theta_2$ and $\xi_B = \theta_3 x + \theta_4$.

The theoretical idea

➜ Following *Pérez-Salinas et al.* procedure, we can build a *universal quantum regressor* for approximating $y = f(x)$. The model can be:

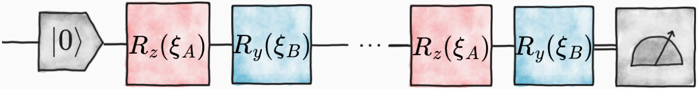


Figure 4: Here $\xi_A = \theta_1 x + \theta_2$ and $\xi_B = \theta_3 x + \theta_4$.

➜ and then use some $E[\hat{O}]$ as predictor:

$$y_{pred} = \langle 0|\mathcal{C}^\dagger(x;\boldsymbol{\theta})\hat{O}\,\mathcal{C}(x;\boldsymbol{\theta})|0\rangle \,. \tag{4}$$

The theoretical idea

➜ Following *Pérez-Salinas et al.* procedure, we can build a *universal quantum regressor* for approximating $y = f(x)$. The model can be:

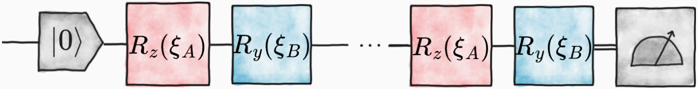


Figure 4: Here $\xi_A = \theta_1 x + \theta_2$ and $\xi_B = \theta_3 x + \theta_4$.

➜ and then use some $E[\hat{O}]$ as predictor:

$$y_{pred} = \langle 0|\mathcal{C}^\dagger(x;\boldsymbol{\theta})\hat{O}\,\mathcal{C}(x;\boldsymbol{\theta})|0\rangle\,. \qquad (4)$$

➜ Using the parameter-shift rule, we can perform a Stochastic Gradient Descent (SGD) on the hardware.

The theoretical idea

➔ Following *Pérez-Salinas et al.* procedure, we can build a *universal quantum regressor* for approximating $y = f(x)$. The model can be:

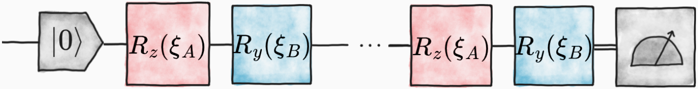


Figure 4: Here $\xi_A = \theta_1 x + \theta_2$ and $\xi_B = \theta_3 x + \theta_4$.

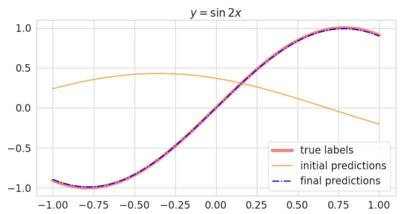
➔ and then use some $E[\hat{O}]$ as predictor:

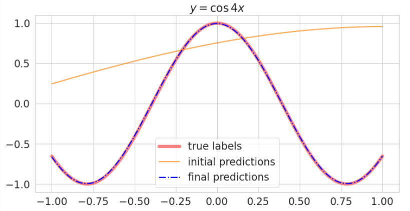
$$y_{pred} = \langle 0|\mathcal{C}^\dagger(x;\boldsymbol{\theta})\hat{O}\,\mathcal{C}(x;\boldsymbol{\theta})|0\rangle\,. \tag{4}$$

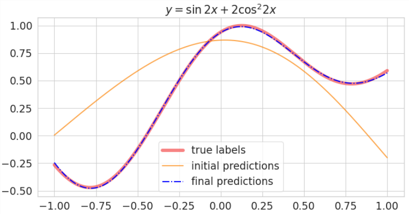
➔ Using the parameter-shift rule, we can perform a Stochastic Gradient Descent (SGD) on the hardware.

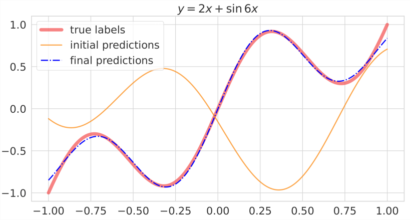
arXiv:2210.10787: *"A quantum analytical Adam descent through parameter shift rule using Qibo."*

Simulation results









Run on the hardware

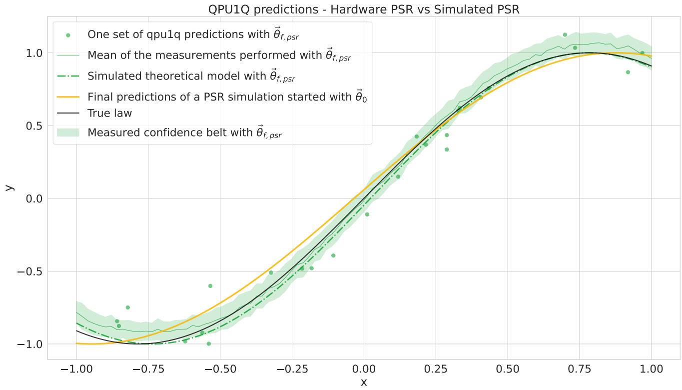


Figure 5: Batch Gradient Descent on the hardware, with gradients evaluated via Parameter-Shift Rule. We take 100 points $\{x_j\}$ in the range $[-1, 1]$ and we make 100 predictions for each x_j. Mean and standard deviation are used for determining the estimations and the confidend belt.

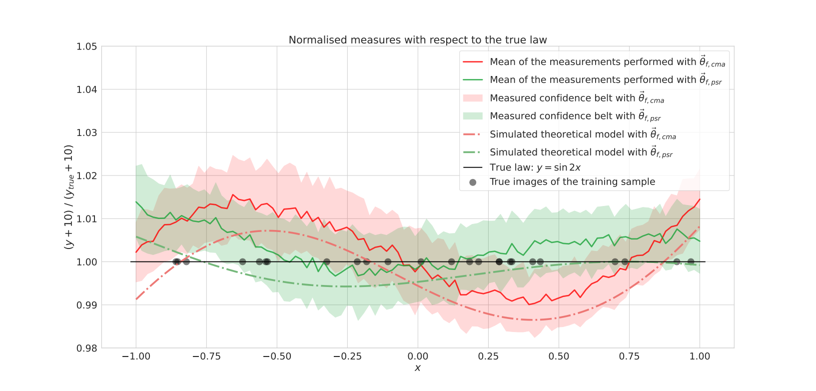


Figure 6: Normalised results of the SGD (green line) compared with true law and a genetic optimizer (red line).

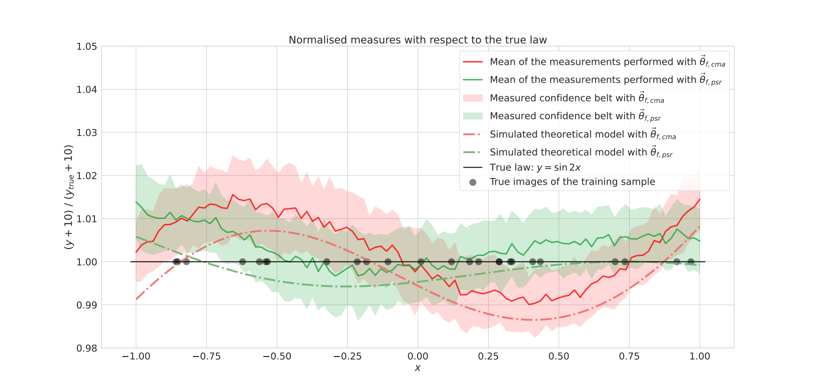


Figure 6: Normalised results of the SGD (green line) compared with true law and a genetic optimizer (red line).

- the full-stack framework works! comparable with a genetic algorithm;

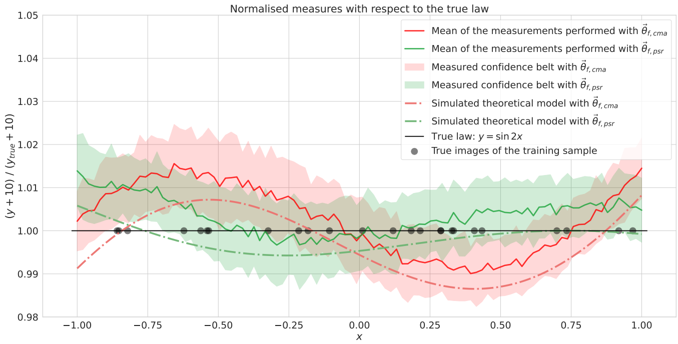


Figure 6: Normalised results of the SGD (green line) compared with true law and a genetic optimizer (red line).

- the full-stack framework works! comparable with a genetic algorithm;
- we can tackle only easy problems: it is slow;

Run on the hardware

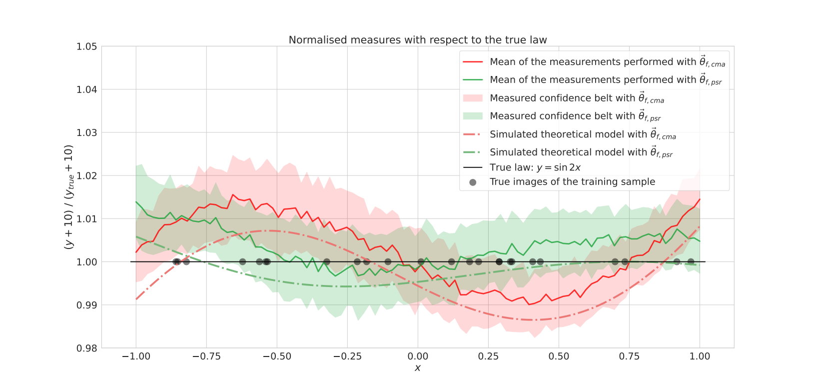


Figure 6: Normalised results of the SGD (green line) compared with true law and a genetic optimizer (red line).

- 👍 the full-stack framework works! comparable with a genetic algorithm;
- 👎 we can tackle only easy problems: it is slow;
- 😐 no mitigation: have been the errors absorbed into the optimization?

✌: how to get noise resistance?

- We want to reproduce the u quark PDF fit of *Pérez-Salinas et al*.

[2] We used Zero Noise Extrapolation (ZNE) and Clifford Data Regresssion (CDR).

- We want to reproduce the *u* quark PDF fit of *Pérez-Salinas et al*.
- We apply error mitigation techniques[2] during a QML training!

[2] We used Zero Noise Extrapolation (ZNE) and Clifford Data Regresssion (CDR).

- We want to reproduce the u quark PDF fit of *Pérez-Salinas et al*.
- We apply error mitigation techniques[2] during a QML training!

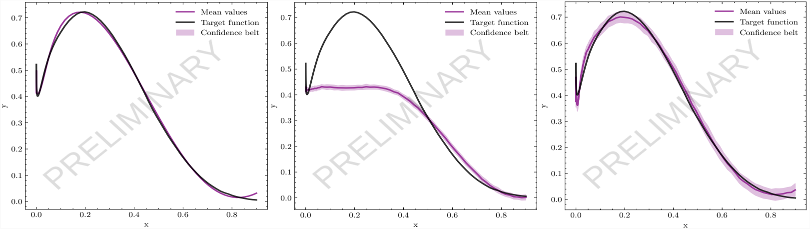


Figure 7: PDF fit performed with different levels of noisy simulation. From left to right, exact simulation, noisy simulation, noisy simulation applying error mitigation to the predictions.

[2] We used Zero Noise Extrapolation (ZNE) and Clifford Data Regresssion (CDR).

- We want to reproduce the u quark PDF fit of *Pérez-Salinas et al*.
- We apply error mitigation techniques[2] during a QML training!

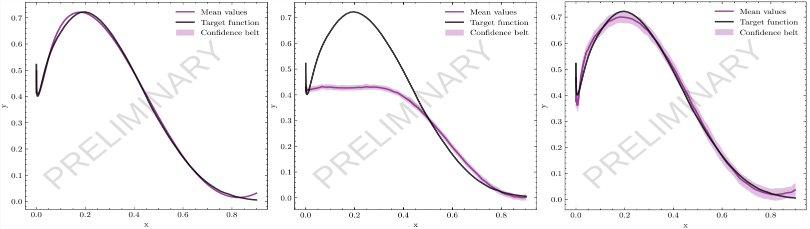


Figure 7: PDF fit performed with different levels of noisy simulation. From left to right, exact simulation, noisy simulation, noisy simulation applying error mitigation to the predictions.

- Run on the hardware upcoming!

[2] We used Zero Noise Extrapolation (ZNE) and Clifford Data Regresssion (CDR).

Conclusions

Conclusions

- I am excited to be part of the `qibo` team and to share it with you:

Conclusions

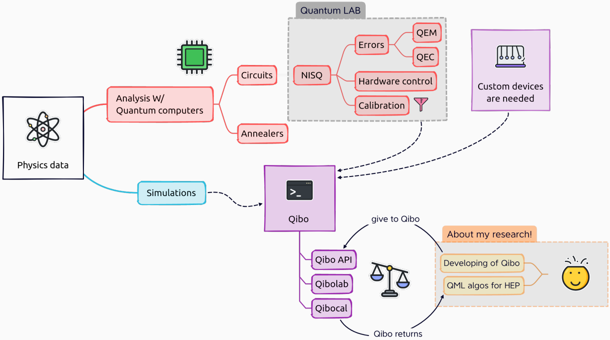
- I am excited to be part of the `qibo` team and to share it with you:
  - it's a perfect environment to tackle QML problems at 360°;

Conclusions

- I am excited to be part of the `qibo` team and to share it with you:
  - it's a perfect environment to tackle QML problems at 360°;
  - is based on a research-centred network, that we would like to grow more and more.

Conclusions

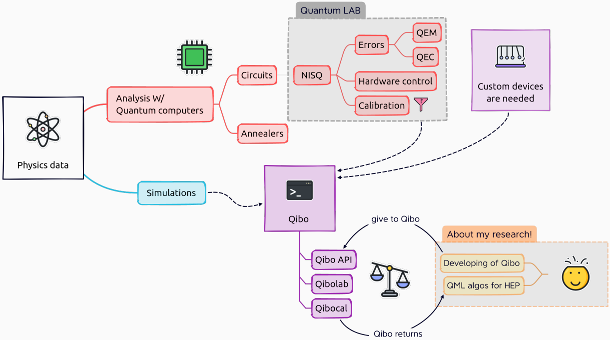
- I am excited to be part of the `qibo` team and to share it with you:
  - it's a perfect environment to tackle QML problems at 360°;
  - is based on a research-centred network, that we would like to grow more and more.



Conclusions

➔ I am excited to be part of the `qibo` team and to share it with you:

- it's a perfect environment to tackle QML problems at 360°;
- is based on a research-centred network, that we would like to grow more and more.



code is open-source here: feel free to make your own contribution!
Have a look to our documentation.