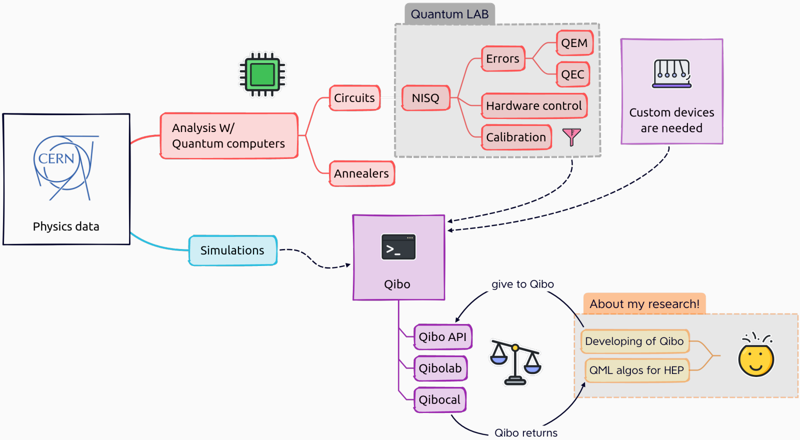
Doing full stack QML using `qibo`

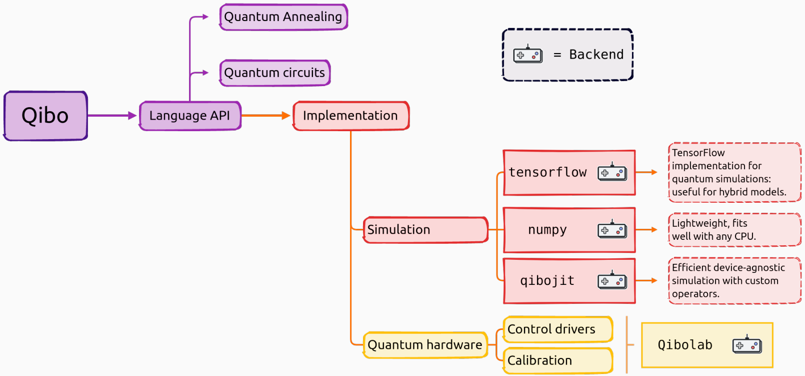
Matteo Robbiati **feat.** Alejandro Sopena

24 May 2023

Working in the NISQ era



The `qibo` ecosystem



Two snapshots

Adiabatic Evolution

- Consider two Hamiltonians H_0 and H_1, whose ground states are $|g_0\rangle$ and $|g_1\rangle$.

Adiabatic Evolution

- Consider two Hamiltonians H_0 and H_1, whose ground states are $|g_0\rangle$ and $|g_1\rangle$.
- We call Adiabatic Evolution the process represented by:

Adiabatic Evolution

- Consider two Hamiltonians H_0 and H_1, whose ground states are $|g_0\rangle$ and $|g_1\rangle$.
- We call Adiabatic Evolution the process represented by:

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

Adiabatic Evolution

- Consider two Hamiltonians H_0 and H_1, whose ground states are $|g_0\rangle$ and $|g_1\rangle$.
- We call Adiabatic Evolution the process represented by:

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

- in which we parametrize the **scheduling** function s.

Adiabatic Evolution

- Consider two Hamiltonians H_0 and H_1, whose ground states are $|g_0\rangle$ and $|g_1\rangle$.
- We call Adiabatic Evolution the process represented by:

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

- in which we parametrize the **scheduling** function s.

```python
# with qibo we can implement an Adiabatic Evolution via trotter formula
from qibo import models, hamiltonians, callbacks

# problem's parameters
nqubits = 1
h0 = hamiltonians.X(nqubits)
h1 = hamiltonians.Z(nqubits)
target_observable = h1

# we track the energy of h1 on the evolved ground state
energies = callbacks.Energy(target_observable)
evolution = models.AdiabaticEvolution(
    h0=h0, h1=h1, s = lambda t : t, dt=0.1, callbacks = [energies])

# calculate the evolved final state at time t=final_time
evolved_state = evolution(final_time = final_time)
```

Quantum Machine Learning - doing ML using QC

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.

Quantum Machine Learning - operating on qubits

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
$\mathcal{M}$, $\mathcal{S}$, $\mathcal{E}$
$\mathcal{U}(\boldsymbol{\theta})\,|q_i\rangle \to |q_f\rangle$

Quantum Machine Learning - natural randomness

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
$\mathcal{M}$, $\mathcal{S}$, $\mathcal{E}$
$\mathcal{U}(\boldsymbol{\theta}) |q_i\rangle \to |q_f\rangle$

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.

VQC execution
$\mathcal{M}, \mathcal{S}, \mathcal{E}$
$\mathcal{U}(\theta)|q_i\rangle \rightarrow |q_f\rangle$

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

Quantum Machine Learning!

Optimizer $\mathcal{O}$

</> hybrid-strategy?

on hardware?

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

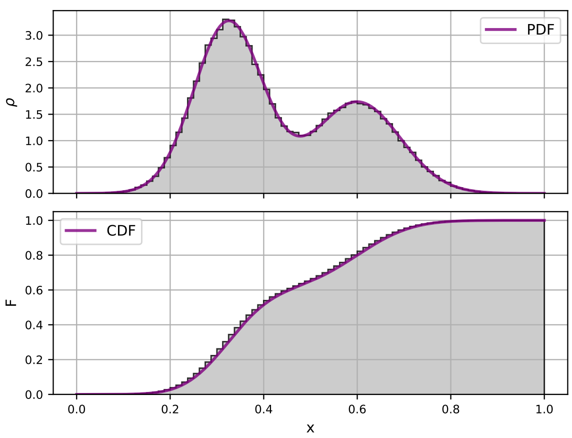
$\mathcal{M}$, $\mathcal{S}$, $\mathcal{E}$

$\mathcal{U}(\boldsymbol{\theta}) |q_i\rangle \rightarrow |q_f\rangle$

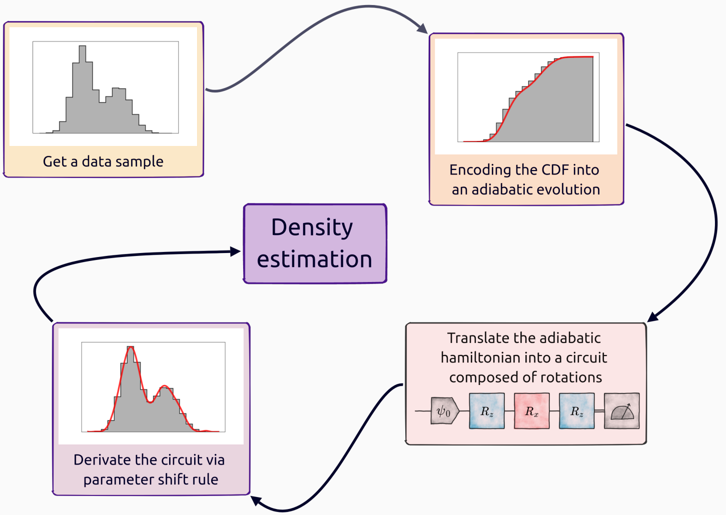
Full stack algorithms using `qibo`

STEP 0: the goal

- Let's consider a sample of data $\{x\}_{k=1}^{N_{\mathrm{data}}}$.
- We can calculate the Cumulative Density Function (CDF) values $F(x)$,
- which are related to the Probability Density Function (PDF) via $\rho(x) = \mathrm{d}F(x)/\mathrm{d}x$.



STEP 0: the idea



Fit CDF with Adiabatic Evolution (AE)

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

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;

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 $(x, F) \to (\tau, E)$.

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 $(x, F) \rightarrow (\tau, E)$.

Training the evolution

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 $(x, F) \to (\tau, E)$.

Training the evolution

1. we run the evolution with random initial $\boldsymbol{\theta}_0$ into the scheduling;

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 $(x, F) \to (\tau, E)$.

Training the evolution

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;

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 $(x, F) \to (\tau, E)$.

Training the evolution

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 $J_{\rm mse}$:

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

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 $(x, F) \to (\tau, E)$.

Training the evolution

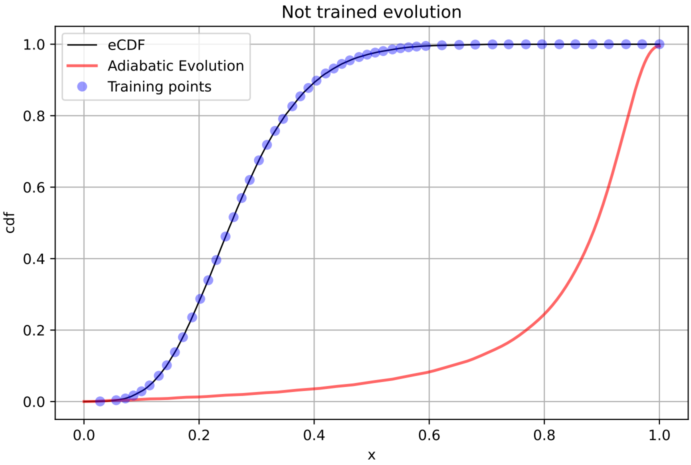
1. we run the evolution with random initial $\boldsymbol{\theta}_0$ into the scheduling;
2. we track the energy of a Pauli Z during the evolution;
3. we calculate a loss function $J_{\rm mse}$:

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

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

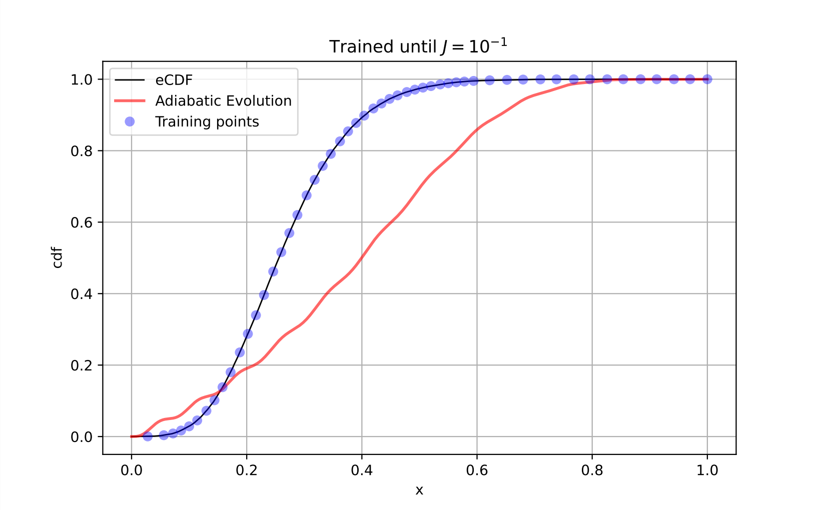
A toy example with `nqubits=1` - starting point

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



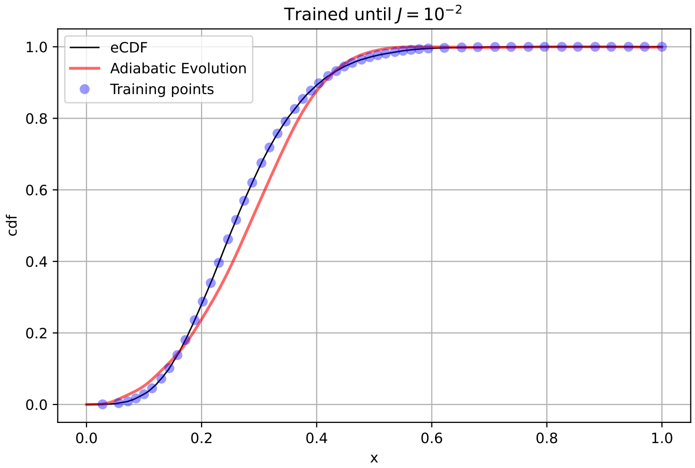
A toy example - until $J_{\mathrm{MSE}} = 10^{-1}$

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



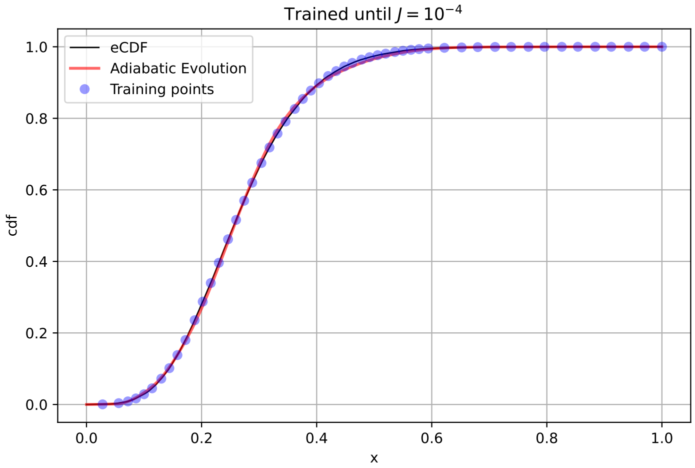
A toy example - until $J_{\text{MSE}} = 10^{-2}$

- `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`



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

From $\{H_{\mathrm{ad}}\}$ to a derivable circuit $\mathcal{C}_R$

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

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

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

From $\{H_{\mathrm{ad}}\}$ to a derivable circuit $\mathcal{C}_R$

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

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

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

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

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

From $\{H_{\mathrm{ad}}\}$ to a derivable circuit $\mathcal{C}_R$

- 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.

From $\{H_{\mathrm{ad}}\}$ to a derivable circuit $\mathcal{C}_R$

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

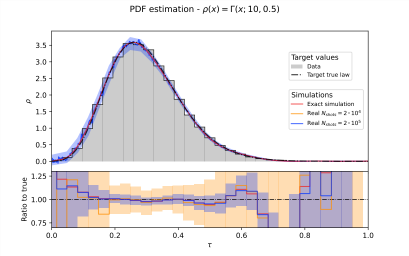
1. translate the Hamiltonians' sequence into a single unitary:

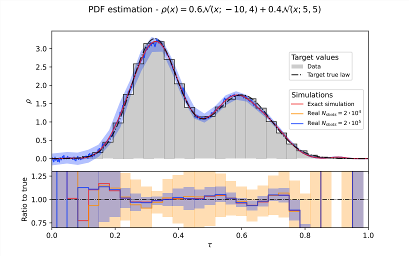
$$\prod_{j=1}^{n} e^{-iH_j \mathrm{d}t} \rightarrow \mathcal{U}(t);$$

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.





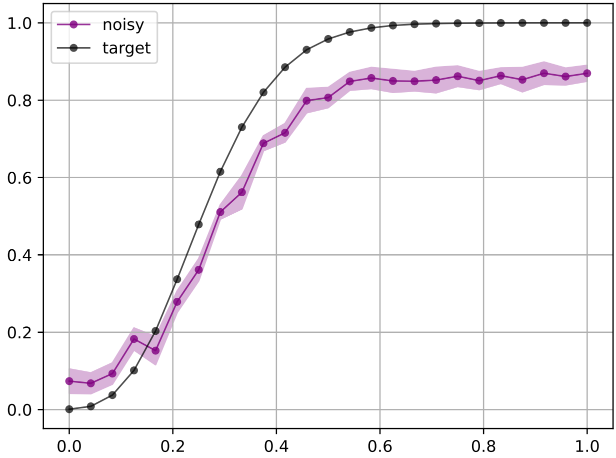


Figure 1: $N_{\text{runs}} = 10$ evaluations of CDF predictions for $N_{\text{data}} = 25$ using $N_{\text{shots}} = 1000$.

Ball to qw5q_gold

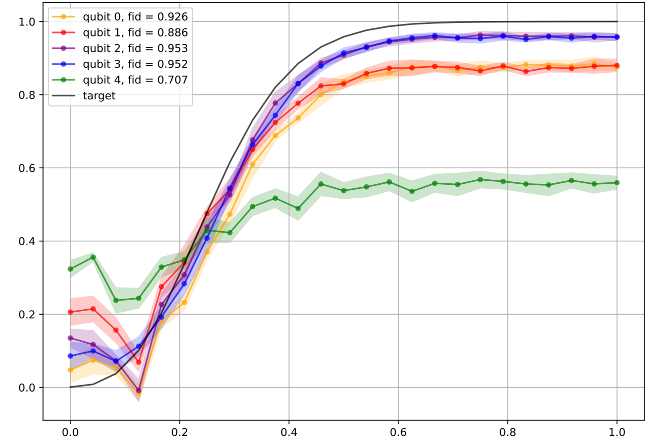


Figure 2: $N_{\text{runs}} = 10$ evaluations of CDF predictions for $N_{\text{data}} = 25$ using $N_{\text{shots}} = 1000$ and each qubit of `qw5q_gold`.

The importance of the `qiboteam`

- These first results open several questions:

The importance of the `qiboteam`

- These first results open several questions:
  - are these errors compatible with transmon SoA?

The importance of the qiboteam

- These first results open several questions:
  - are these errors compatible with transmon SoA?
  - which relationship between my results' error and hardware ones?

The importance of the `qiboteam`

- These first results open several questions:
  - are these errors compatible with transmon SoA?
  - which relationship between my results' error and hardware ones?
  - where are the execution time bottlenecks?

The importance of the `qiboteam`

- These first results open several questions:
  - are these errors compatible with transmon SoA?
  - which relationship between my results' error and hardware ones?
  - where are the execution time bottlenecks?
  - what if we train the entire process on the hardware?

The importance of the `qiboteam`

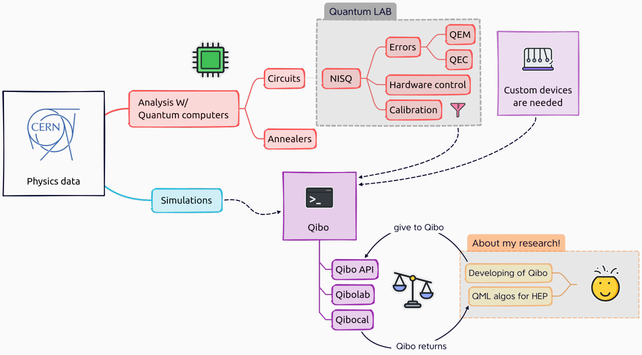
- These first results open several questions:
  - are these errors compatible with transmon SoA?
  - which relationship between my results' error and hardware ones?
  - where are the execution time bottlenecks?
  - what if we train the entire process on the hardware?
  - what if we apply error mitigation on predictions?

The importance of the `qiboteam`

- These first results open several questions:
  - are these errors compatible with transmon SoA?
  - which relationship between my results' error and hardware ones?
  - where are the execution time bottlenecks?
  - what if we train the entire process on the hardware?
  - what if we apply error mitigation on predictions?
  - what if we apply real-time error mitigation?

The importance of the `qiboteam`

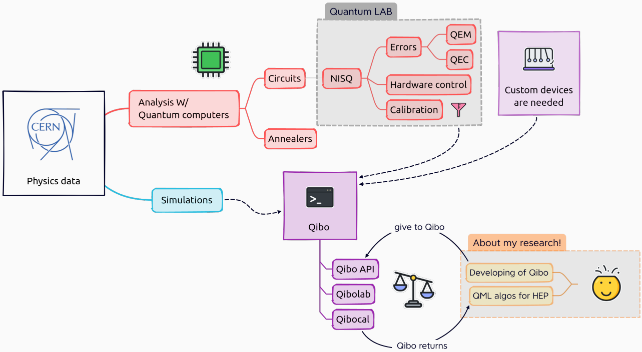
- These first results open several questions:
  - are these errors compatible with transmon SoA?
  - which relationship between my results' error and hardware ones?
  - where are the execution time bottlenecks?
  - what if we train the entire process on the hardware?
  - what if we apply error mitigation on predictions?
  - what if we apply real-time error mitigation?



The importance of the `qiboteam`

➔ These first results open several questions:

- are these errors compatible with transmon SoA?
- which relationship between my results' error and hardware ones?
- where are the execution time bottlenecks?
- **what if we train the entire process on the hardware?**
- what if we apply error mitigation on predictions?
- what if we apply real-time error mitigation?



A QML training on `tii1q_b1`

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

A QML training on `tii1q_b1`

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

A QML training on `tii1q_b1`

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

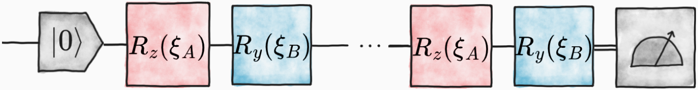


Figure 3: Here $\xi_A = \theta_1 x + \theta_2$ and $\xi_B = \theta_3 x + \theta_4$.

A QML training on `tii1q_b1`

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

➜ The model can be:

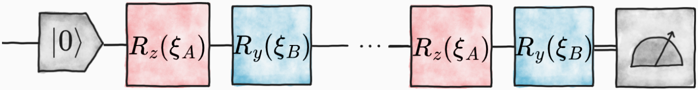


Figure 3: Here $\xi_A = \theta_1 x + \theta_2$ and $\xi_B = \theta_3 x + \theta_4$.

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

$$y_{pred} = \langle 0|\mathcal{C}^\dagger(x;\boldsymbol{\theta})\hat{O}\,\mathcal{C}(x;\boldsymbol{\theta})|0\rangle\,. \tag{2}$$

A QML training on `tii1q_b1`

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

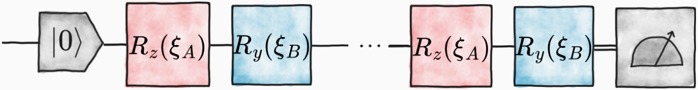


Figure 3: Here $\xi_A = \theta_1 x + \theta_2$ and $\xi_B = \theta_3 x + \theta_4$.

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

$$y_{pred} = \langle 0|\mathcal{C}^\dagger(x;\boldsymbol{\theta})\hat{O}\,\mathcal{C}(x;\boldsymbol{\theta})|0\rangle\,. \tag{2}$$

- Using the parameter-shift rule, we can perform a gradient descent on `tii1q_b1`.

Running on `tii1q_b1`

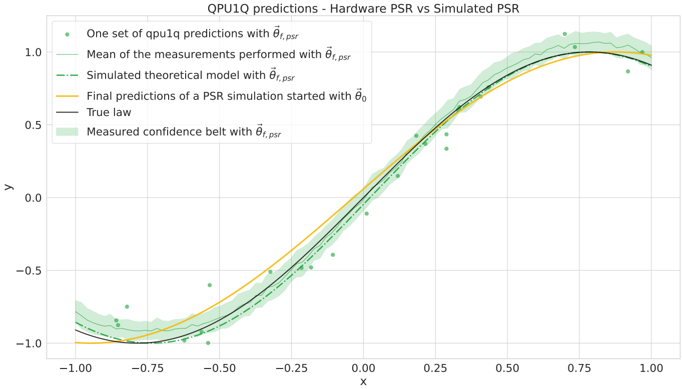


Figure 4: Batch Gradient Descent on the hardware, with gradients evaluated via Parameter-Shift Rule. We take 100 points $\{x_j\}$ in the range $[-1, 1]$ and we make 100 predictions for each x_j. Mean and standard deviation are used for determining the estimations and the confidend belt.

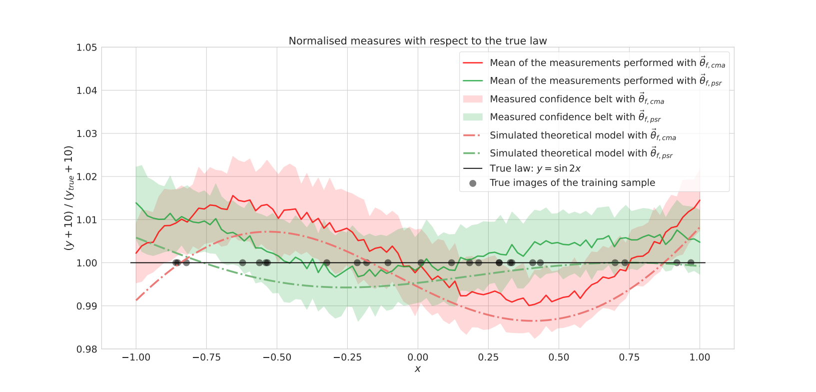


Figure 5: Normalised results of the SGD (green line) compared with true law and a genetic optimizer (red line).

Normalised results

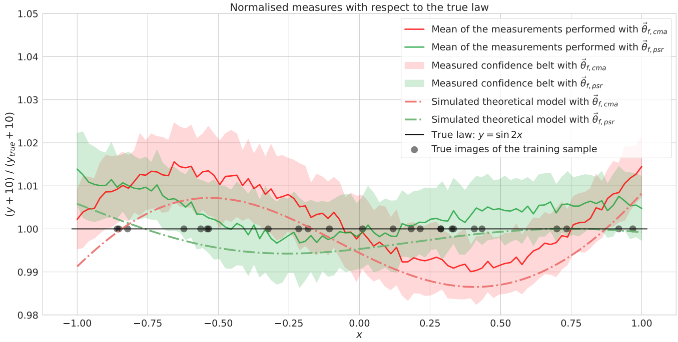


Figure 5: Normalised results of the SGD (green line) compared with true law and a genetic optimizer (red line).

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

Normalised results

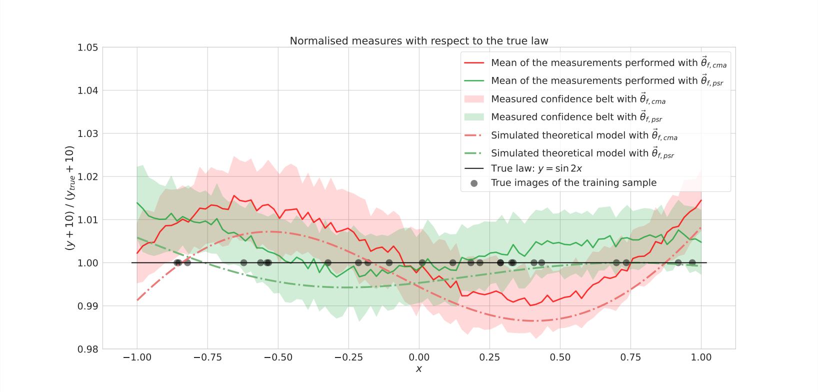


Figure 5: Normalised results of the SGD (green line) compared with true law and a genetic optimizer (red line).

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

Normalised results

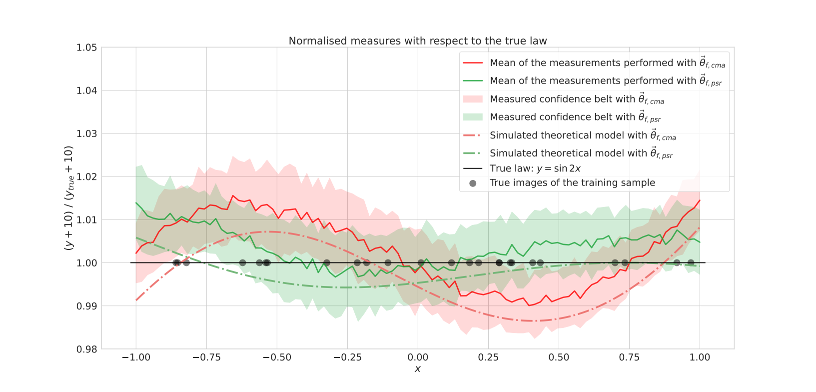


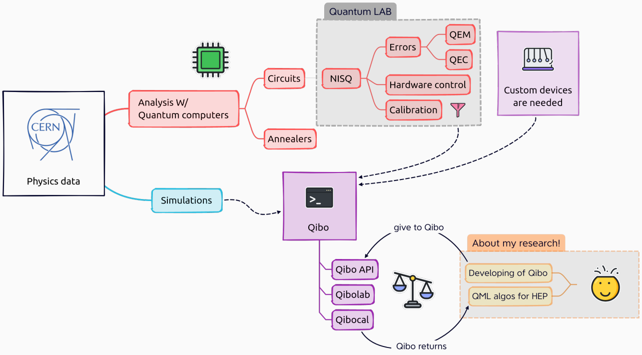
Figure 5: Normalised results of the SGD (green line) compared with true law and a genetic optimizer (red line).

- 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?

The importance of the `qiboteam`

➔ These first results open several questions:

- are these errors compatible with transmon SoA?
- which relationship between my results' error and hardware ones?
- where are the execution time bottlenecks?
- what if we train the entire process on the hardware?
- **what if we apply error mitigation on predictions?**
- what if we apply real-time error mitigation?



Mitigated prediction of the PDF

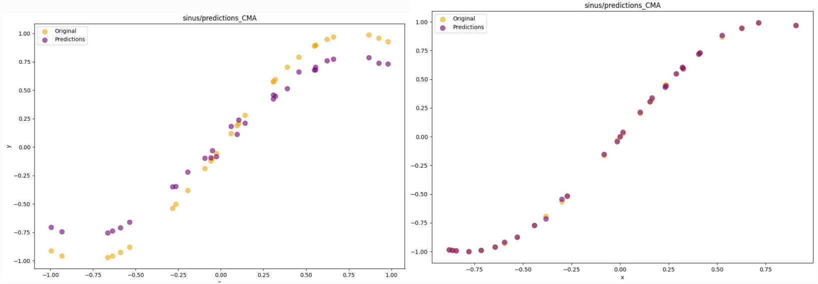


Figure 6: Predictions of a sinus function performed on `tii1q_b1` without and with CDR mitigation.

- These results open to new questions!

Mitigated prediction of the PDF

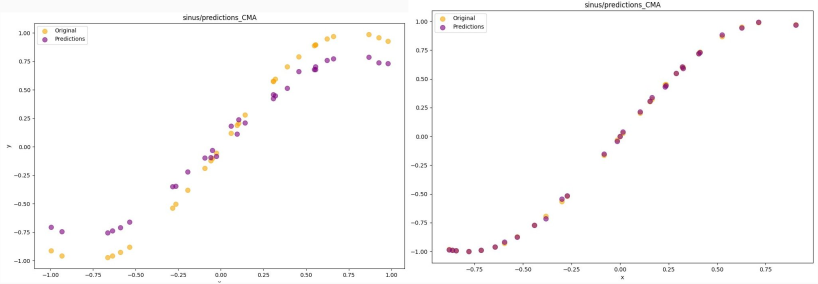


Figure 6: Predictions of a sinus function performed on `tii1q_b1` without and with CDR mitigation.

➔ These results open to new questions!

- can QEM help the training with a faster convergence?

Mitigated prediction of the PDF

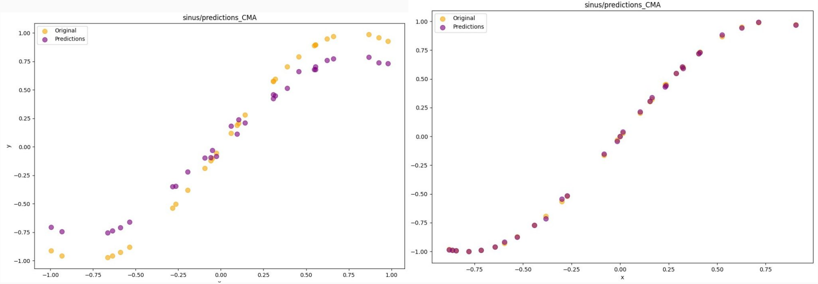


Figure 6: Predictions of a sinus function performed on `tii1q_b1` without and with CDR mitigation.

➔ These results open to new questions!

- can QEM help the training with a faster convergence?
- what's the right balance between error absorption and QEM?

Mitigated prediction of the PDF

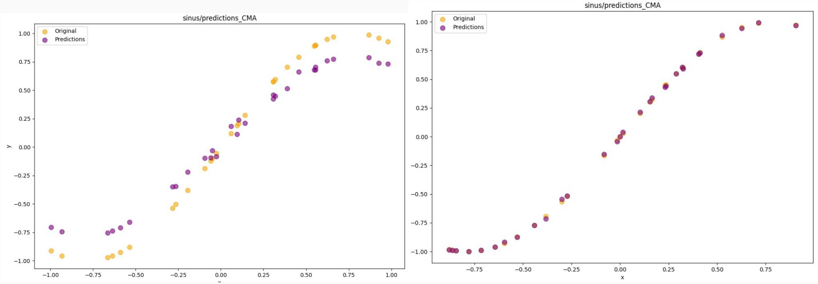


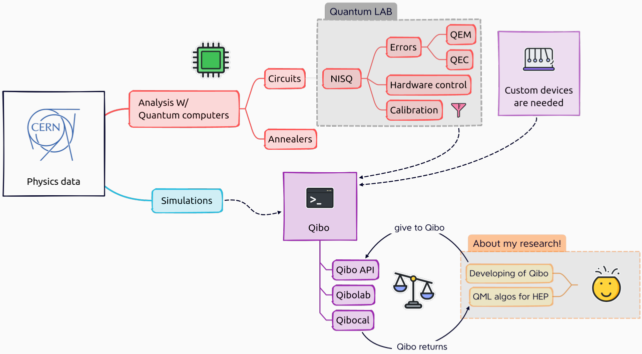
Figure 6: Predictions of a sinus function performed on `tii1q_b1` without and with CDR mitigation.

➔ These results open to new questions!

- can QEM help the training with a faster convergence?
- what's the right balance between error absorption and QEM?
- can we train the unmitigated hardware to be self-resistent?

The importance of the `qiboteam`

- These first results open several questions:
  - are these errors compatible with transmon SoA?
  - which relationship between my results' error and hardware ones?
  - where are the execution time bottlenecks?
  - what if we train the entire process on the hardware?
  - what if we apply error mitigation on predictions?
  - **what if we apply real-time error mitigation?**



Real-time error mitigation

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

[1] 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[1] during a QML training!

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

Real-time error mitigation

- We want to reproduce the u quark PDF fit of *Pérez-Salinas et al*.
- We apply error mitigation techniques[1] during a QML training!

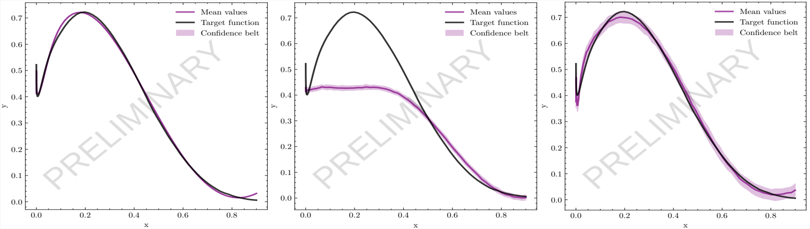


Figure 7: PDF fit performed with different levels of noisy simulation. From left to right, exact simulation, noisy simulation, noisy simulation applying error mitigation to the predictions.

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

Real-time error mitigation

- We want to reproduce the *u* quark PDF fit of *Pérez-Salinas et al*.
- We apply error mitigation techniques[1] during a QML training!

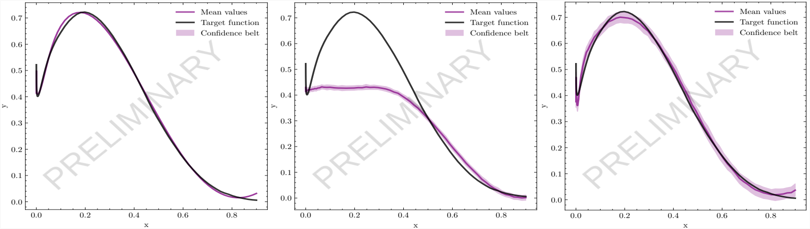


Figure 7: PDF fit performed with different levels of noisy simulation. From left to right, exact simulation, noisy simulation, noisy simulation applying error mitigation to the predictions.

- **But on the hardware?**

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

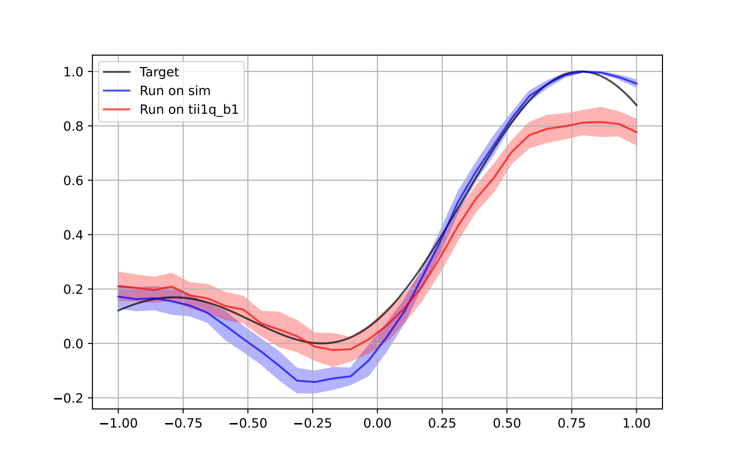


Figure 8: Full training is here performed on `tii1q_b1`. We then used the $\boldsymbol{\theta}_{\text{best}}$ to make statistics on 30 points and 30 runs with simulation (blue line) and on the device (red line).

Conclusions

qibo as research network!

`qibo` as research network!

- The `qibo` environment is perfect for this kind of research, because

`qibo` as research network!

- The `qibo` environment is perfect for this kind of research, because
  - we have resources deputed to the research;

qibo as research network!

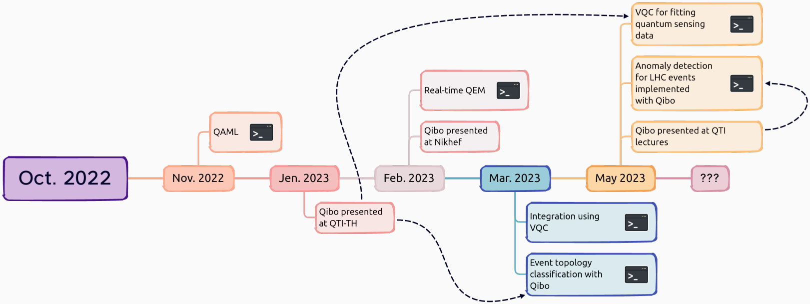
- The qibo environment is perfect for this kind of research, because
  - we have resources deputed to the research;
  - we are a fully connected research network.

qibo as research network!

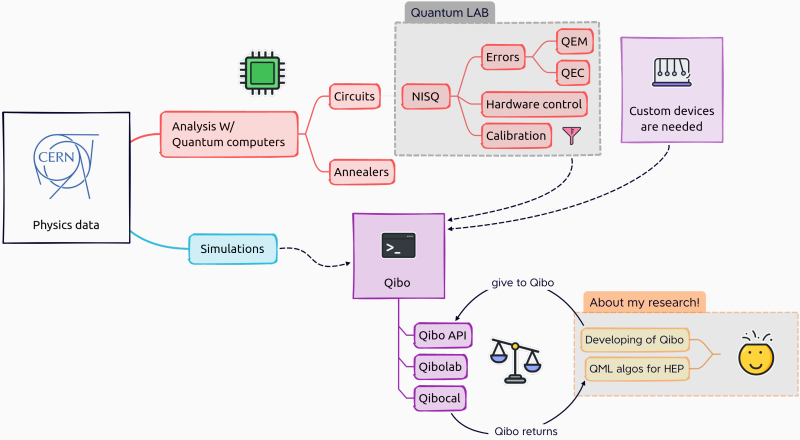
- The qibo environment is perfect for this kind of research, because
  - we have resources deputed to the research;
  - we are a fully connected research network.
- Moreover, qibo is growing as international environment:

`qibo` as research network!

- The `qibo` environment is perfect for this kind of research, because
  - we have resources deputed to the research;
  - we are a fully connected research network.
- Moreover, `qibo` is growing as international environment:



Thanks!



CERN
Physics data
Analysis W/
Quantum computers
Circuits
Annealers
Quantum LAB
NISQ
Errors
QEM
QEC
Hardware control
Calibration
Custom devices
are needed
Simulations
Qibo
Qibo API
Qibolab
Qibocal
give to Qibo
Qibo returns
About my research!
Developing of Qibo
QML algos for HEP