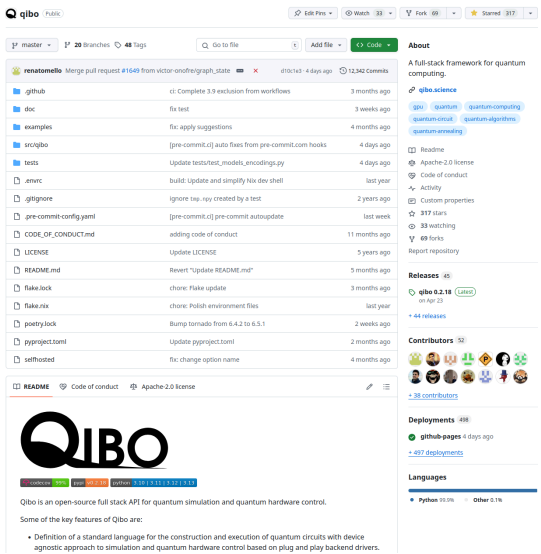
Full-stack quantum machine learning on hybrid quantum-classical platforms

Andrea Papaluca
andrea.papaluca@unimi.it

UNIVERSITÀ
DEGLI STUDI
DI MILANO

QIBO

A worldwide open source collaboration



"Qibo: a framework for quantum simulation with hardware acceleration"

Efthymiou et al, 2020.

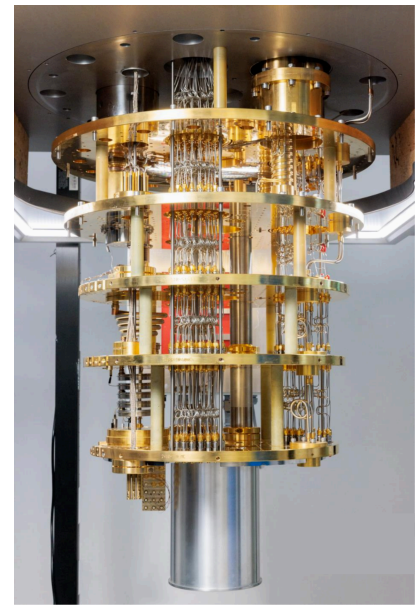
The Quantum Computer

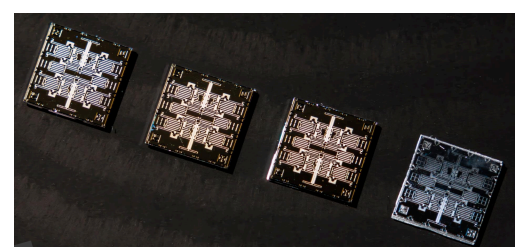
CQT

SG NSCC SINGAPORE National Supercomputing Centre

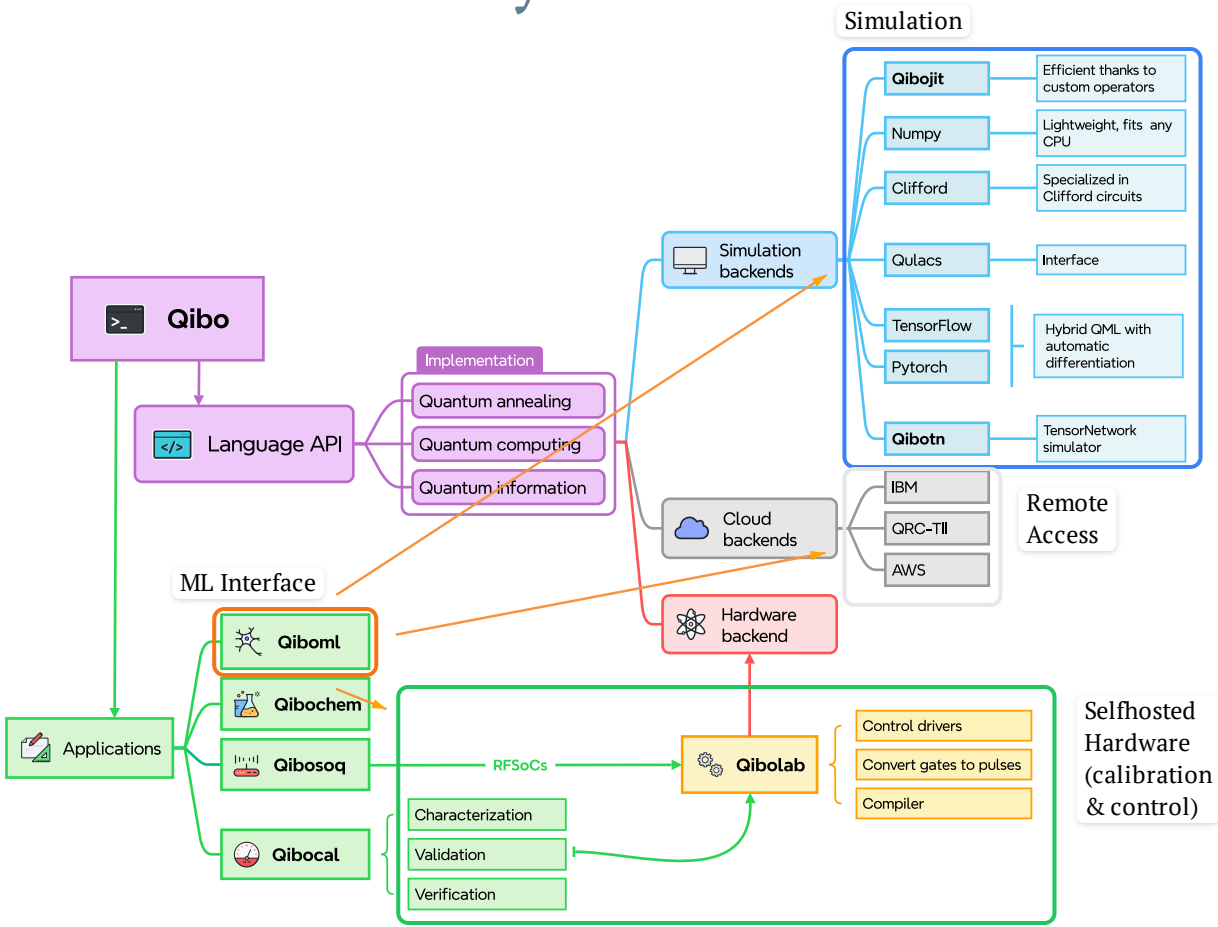
Agency for Science, Technology and Research SINGAPORE

NATIONAL QUANTUM COMPUTING HUB SINGAPORE

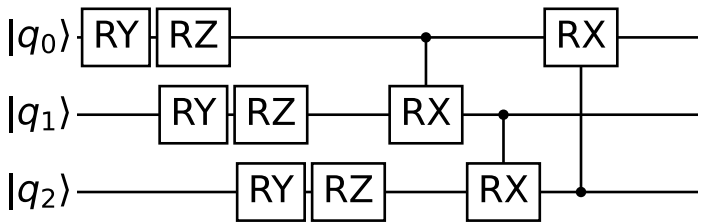


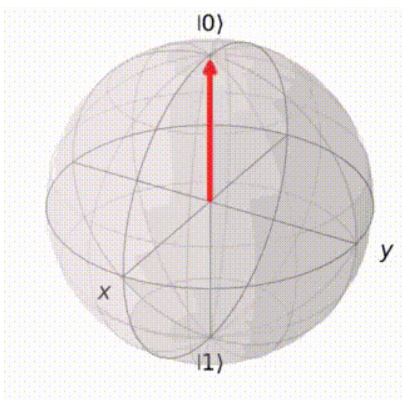


A flexible full-stack ecosystem



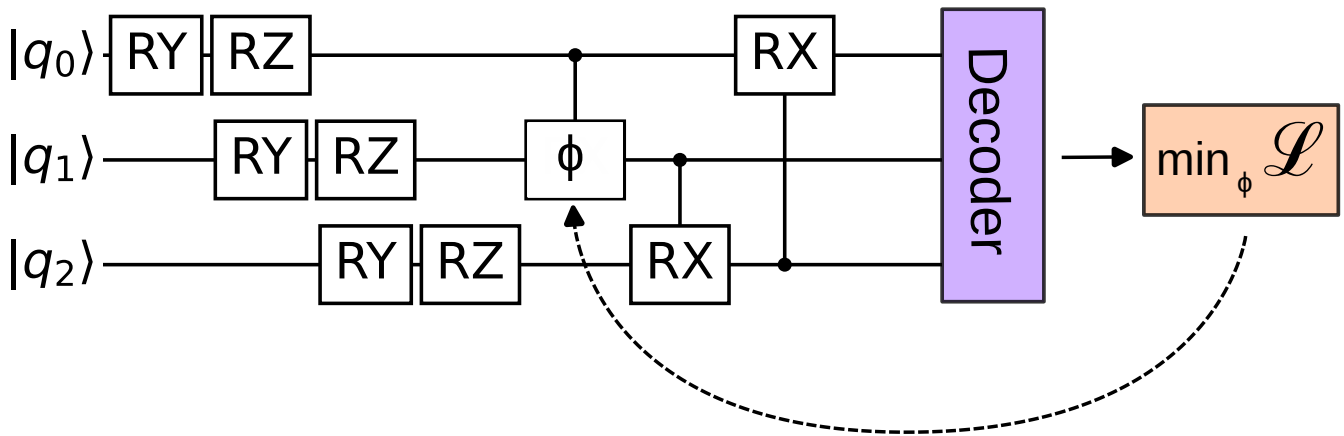
Quantum Computation through circuits





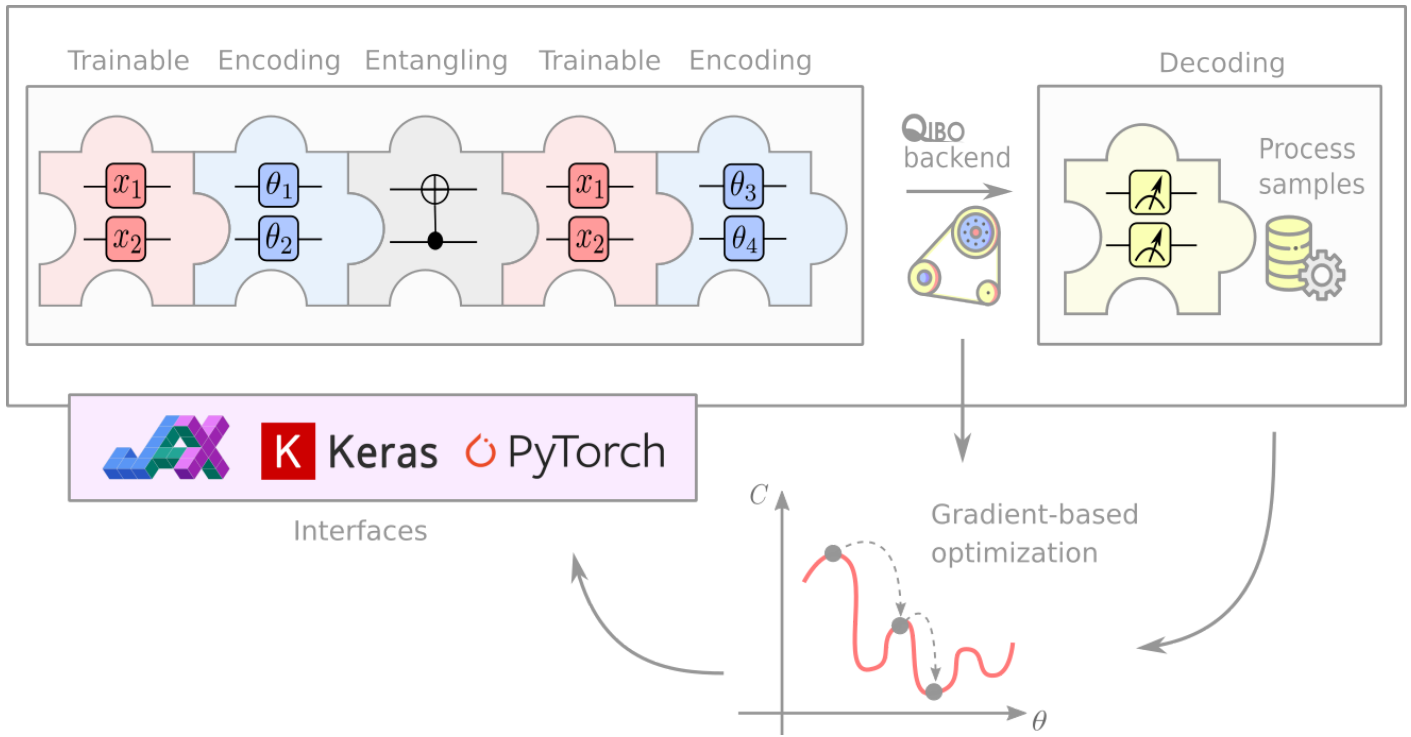

```python
import numpy as np
from qibo import Circuit, gates

# Construct the circuit
circuit  = Circuit(3)
# Add the gates
for q in range(3):
  circuit.add(gates.RY(q=q, theta=np.random.randn()))
  circuit.add(gates.RZ(q=q, theta=np.random.randn()))
  for q in range(3):
    circuit.add(gates.CRX(
      q%3, (q+1)%3, theta=np.random.randn())
    )
# Execute the circuit and get the final state
result = circuit()
print(result.state())
```

Quantum Machine Learning (QML)



- Decode the quantum information contained in the final state
- Compute the loss
- Update the gates' parameters to minimize the loss

The Qiboml pipeline



QML models made easy

Interface agnostic

```
from qiboml.models.encoding import PhaseEncoding
from qiboml.models.decoding import Expectation

# Encoding layer
encoding = PhaseEncoding(3, encoding_gate=gates.RX)
structure = []
# Alternate encoding and trainable layers
for _ in range(5):
    structure.extend([encoding, circuit.copy()])
# Decoding layer
decoding = Expectation(3)
# Prepare some data
x = np.random.uniform(
  0, 2, 300
  ).reshape(100, 3) * np.pi
y = np.sin(x).sum(-1)
```

Keras interface

```
from qiboml.interfaces.keras import QuantumModel
import tensorflow as tf

# This is a keras.Model
q_model = QuantumModel(structure, decoding=decoding)
# you can train it as you normally do for keras
# models
q_model.compile("adam", "mean_squared_error")
q_model.fit(
  tf.convert_to_tensor(x),
  tf.convert_to_tensor(y),
  batch_size=1,
  epochs=10
)
```

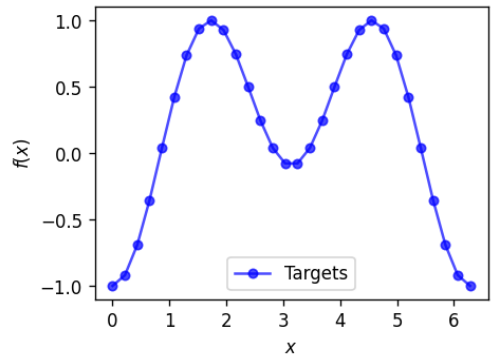
Deploy on any device: even selfhosted QPUs!

- *Just In Time* (JIT) CPU
- GPUs
- Cloud Providers
- Selfhosted QPUs

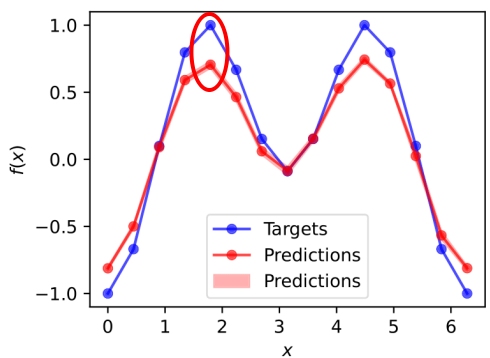

```python
from qibo import set_backend

# Use your selfhosted device through qibolab!
set_backend("qibolab", platform="my_local_platform")
```

Fitting functions on Quantum Hardware

```python
# Target function
def f(x):
    return torch.sin(x) ** 2 - 0.3 * torch.cos(x)
# Training Dataset
num_samples = 30
x_train = torch.linspace(
  0, 2 * np.pi, num_samples
  ).unsqueeze(1)
y_train = f(x_train)
```



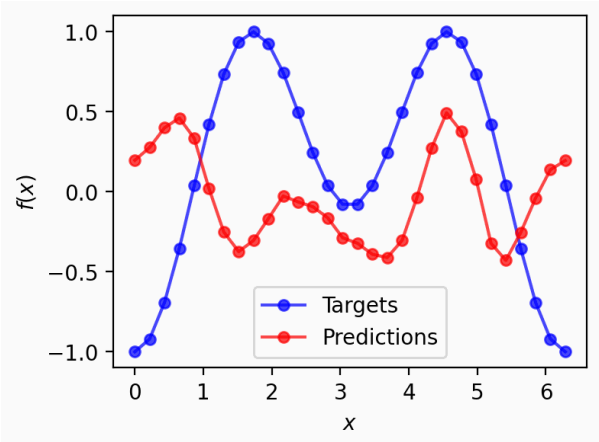

```python
# Using qibolab at NQCH
set_backend("qibolab", platform="sinq-20")
# Define the transpiler
glist = [gates.GPI2, gates.RZ, gates.Z, gates.CZ]
natives = NativeGates(0).from_gatelist(glist)
transpiler = Passes([Unroller(natives)])
# Decoding layer
decoding = Expectation(
  nqubits=1,
  wire_names=[19], # select the qubit to use
  nshots=1024, # set the number of shots
  transpiler=transpiler
)
```



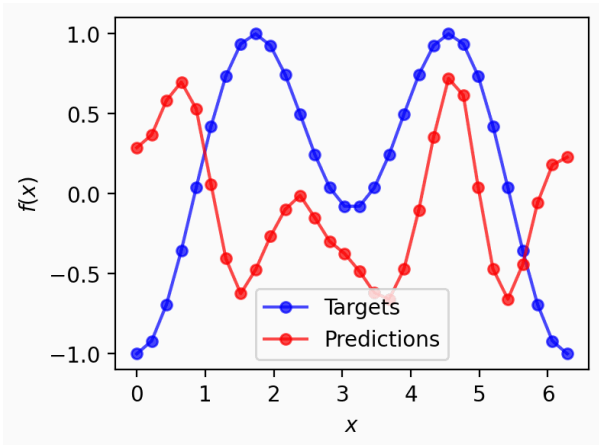
NATIONAL QUANTUM COMPUTING HUB
SINGAPORE

Mitigating Errors

Simulated Pauli Noise

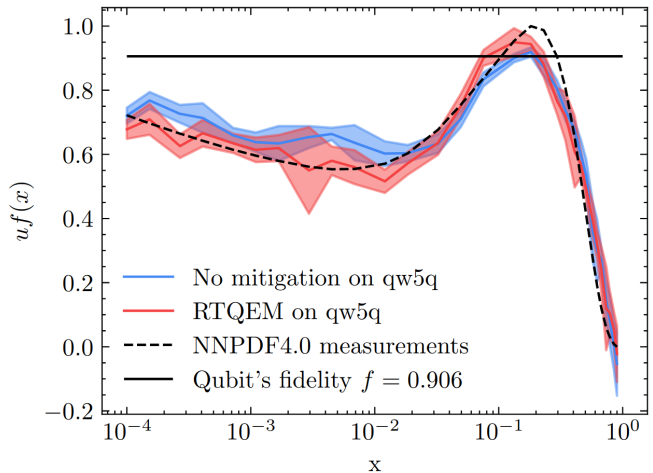


CDR mitigation




```
decoding = Expectation(
    nshots=5000,
    nqubits=1,
    mitigation_config={
    "real_time": True,
    "method": "CDR",
    "method_kwargs": {"n_training_samples": 50}
  }
)
```

Surpassing qubit fidelity

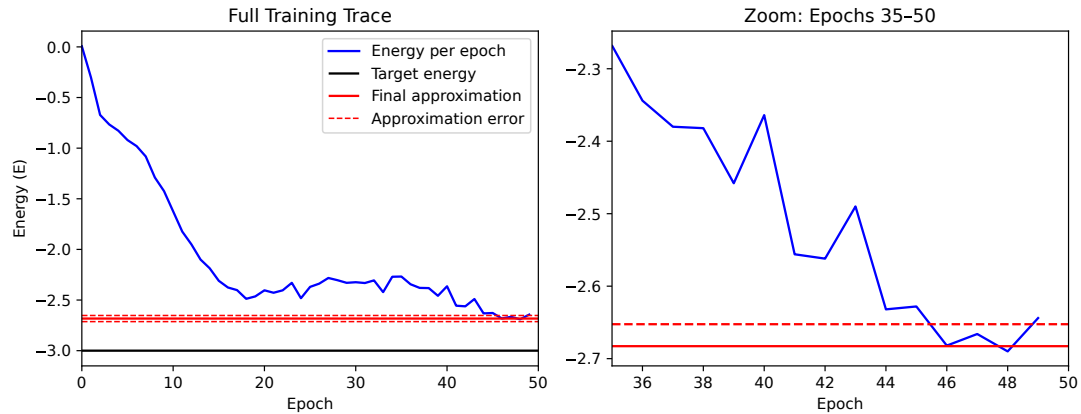


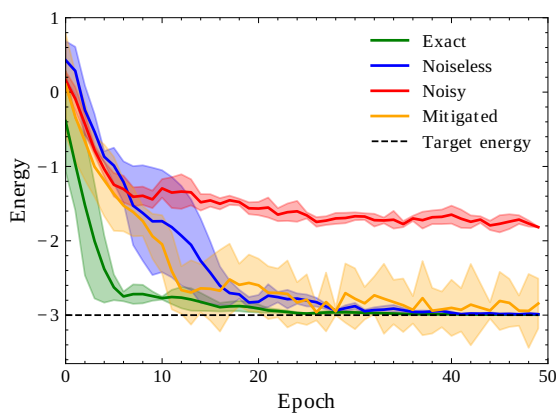
"Real-time error mitigation for variational optimization on quantum hardware"

Robbiati et al.

Finding ground states through VQEs

```
# by default this computes:
# <H> = <Z0 + Z1 + Z2>
# any observable can be used though
decoder = Expectation(
  nqubits=3,
  wire_names=[8, 3, 13],
  nshots=1000,
  transpiler=transpiler
  )
circuit = HardwareEfficient(3, nlayers=2)
model = QuantumModel([circuit,], decoder)
optimizer = torch.optim.Adam(
  model.parameters(),
  lr=0.25
  )
for epoch in range(50):
  optimizer.zero_grad()
  cost = model()
  cost.backward()
  optimizer.step()
```



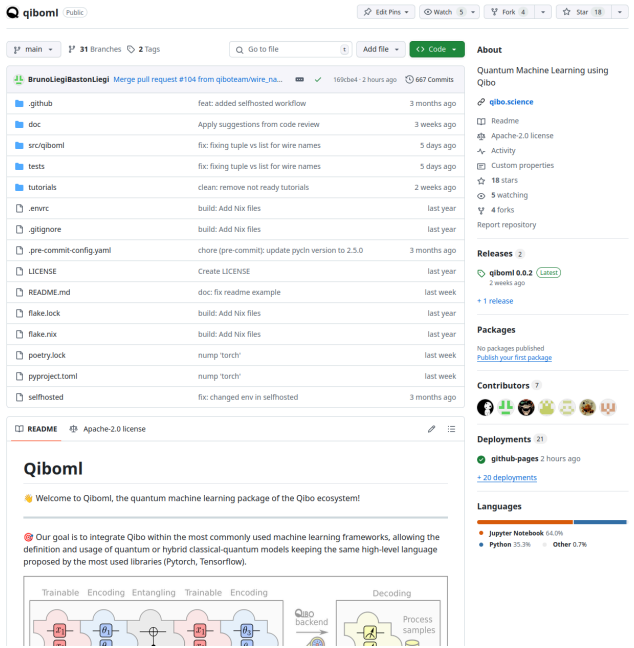


NATIONAL QUANTUM COMPUTING HUB SINGAPORE

(Simulation)

Accelerating QML research

- Qiboml: an opensource library for QML applications
- Interfaces Qibo to popular ML frameworks (torch, keras, jax/flax?)
- Deploy on any Qibo-compatible backend:
  - *jit* CPU
  - GPU
  - cloud
  - selfhosted QPU!
  - Tensor Network (quimb and matchatea)?
- Mitigate errors in real time (RTQEM)!



Thank you for your attention!