QIBO

Quantum simulation with hardware acceleration (arXiv:2009.01845)

Stefano Carrazza

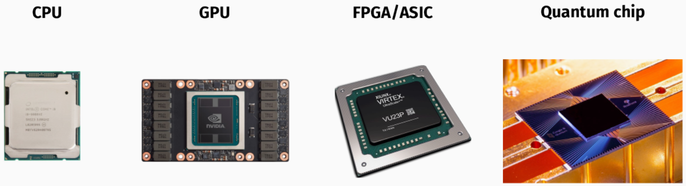
25th September 2020, Center for Quantum Technologies (CQT), Singapore.

Università degli Studi di Milano, INFN Milan, TII

Introducing Qibo

Introduction

From a practical point of view, we are moving towards new technologies, in particular hardware accelerators:



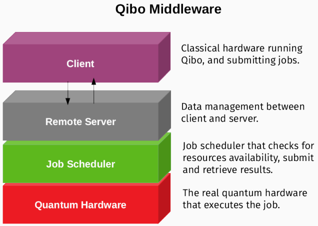
Moving from general purpose devices $\Rightarrow$ application specific

Context

Qibo is the open-source API for a new quantum hardware developed at:

$\Rightarrow$ **Barcelona** by UB, BSC, IFAE, QQT

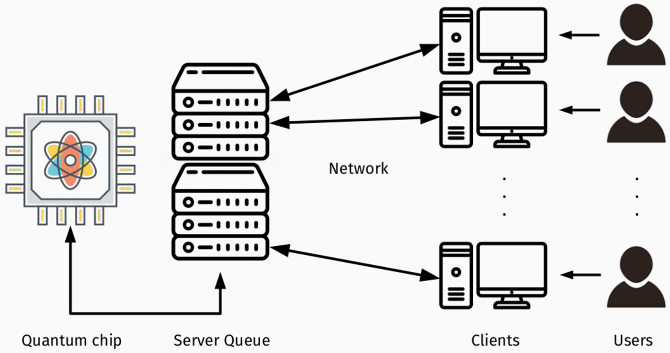
$\Rightarrow$ **Abu Dhabi** by TII


Qibo Middleware
Client
Classical hardware running Qibo, and submitting jobs.
Remote Server
Data management between client and server.
Job Scheduler
Job scheduler that checks for resources availability, submit and retrieve results.
Quantum Hardware
The real quantum hardware that executes the job.

Expected machines based on different technologies for multiple qubits.

Motivation

Why a quantum middleware?



Natural questions:

1. How to prepare and execute quantum algorithms?
2. How to make quantum hardware accessible to users?

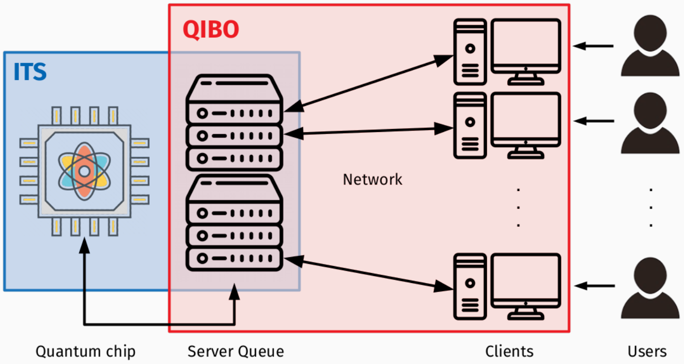
The middleware definition

Computing framework

Development of a quantum computing framework which encodes quantum algorithms in a programming API.

Infrastructual setup

Development of an IT infrastructure for users to execute and retrieve results from quantum hardware using Qibo.

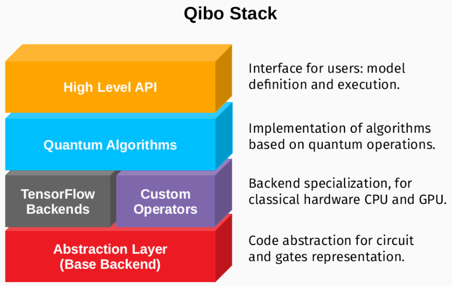


The Qibo framework

Qibo module design

Qibo is a general purpose quantum computing **API** specialized in:

- Model simulation on classical hardware: CPUs and GPUs
- Model execution on quantum hardware

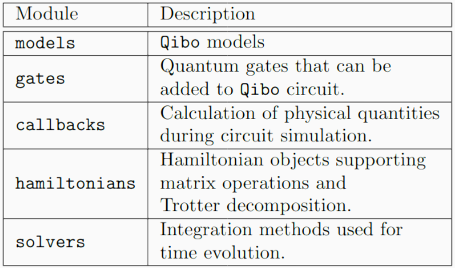

Qibo Stack
High Level API
Quantum Algorithms
TensorFlow Backends
Custom Operators
Abstraction Layer (Base Backend)
Interface for users: model definition and execution.
Implementation of algorithms based on quantum operations.
Backend specialization, for classical hardware CPU and GPU.
Code abstraction for circuit and gates representation.

Furthermore Qibo provides the possibility to:

- create a codebase for quantum algorithms
- mix classical and quantum algorithms

Qibo modules

Modules supported by **Qibo 0.1.0**:

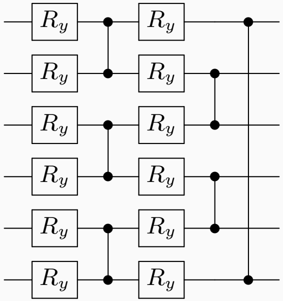


Module	Description
`models`	`Qibo` models
`gates`	Quantum gates that can be added to `Qibo` circuit.
`callbacks`	Calculation of physical quantities during circuit simulation.
`hamiltonians`	Hamiltonian objects supporting matrix operations and Trotter decomposition.
`solvers`	Integration methods used for time evolution.

Modules are designed to work on simulation and quantum hardware.

Qibo 0.1.0 main features

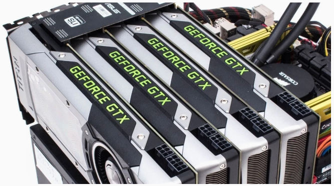
- **Circuit-based quantum processors**
  - State wave-function propagation
  - Controlled gates
  - Measurements
  - Density matrices and noise
  - Callbacks
  - Gate Fusion
  - Distributed computation
  - Variational Quantum Eigensolver
- **Annealing quantum processors**
  - Time evolution of quantum states
  - Adiabatic Evolution simulation
  - Scheduling determination
  - Trotter decomposition
  - QAOA



$$i\hbar\frac{\partial}{\partial t}|\psi(t)\rangle = H(s)|\psi(t)\rangle$$

Qibo technical aspects

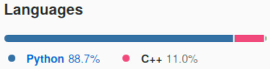
1. efficient simulation engine for:
   - multithreading CPU
   - single-GPU
   - multi-GPU
2. designed with modern standards:
   - installers (`pip install qibo`)
   - documentation
   - unit testing
   - continuous integration
3. released as an open-source code
   `https://qibo.science`



Qibo language and technologies

Project statistics:

10'000 lines of code in `Python/C++`.



The current simulation engine is based on:

- **TensorFlow 2:**
  - Representation of quantum states, density matrices and gates.
  - Optimizes linear algebra operations on CPU/GPU.
  - Introduces an abstraction interface to hardware implementation.
  - Warning: requires custom operators and fine tuning for efficiency.
- **Numpy/Scipy:** linear algebra object definition and optimizers.
- **Joblib:** manages the computation distribution on **multi-GPU**.

Circuit simulation with Qibo

Quantum circuit simulation

`Qibo` simulates the behaviour of quantum circuits using dense complex state vectors $\psi(\sigma_1, \sigma_2, \dots, \sigma_N) \in \mathbb{C}$ in the computational basis where $\sigma_i \in \{0, 1\}$ and N is the total number of qubits in the circuit.

The final state of circuit evaluation is given by:

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

where the sum runs over qubits targeted by the gate.

- $G(\boldsymbol{\sigma}, \boldsymbol{\sigma'})$ is a gate matrix which acts on the state vector.
- $\psi(\boldsymbol{\sigma})$ from a simulation point of view is bounded by memory.

Some useful quantum gates

Rotations around the axis of the Bloch sphere:

$$R_x(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix},\, R_z(\theta) = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix}$$

The **controlled-phase gate** and **Hadamard**:

$$C_z = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix},\, H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

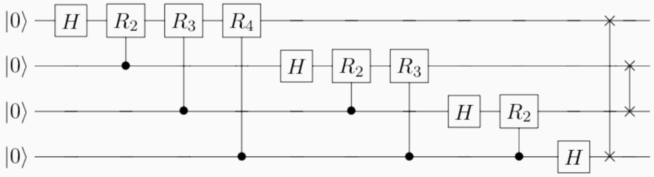
Others examples are: Pauli X/Y/Z, Toffoli, Identity, Controlled-Not.

Quantum Fourier Transform

- The **QFT** is defined as:

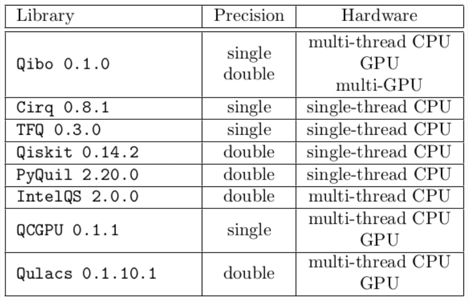
$$|x\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} w_N^{xk} |k\rangle$$

- The QFT can be represented by the **circuit design**:



Benchmark configuration

We benchmark Qibo with the following libraries:

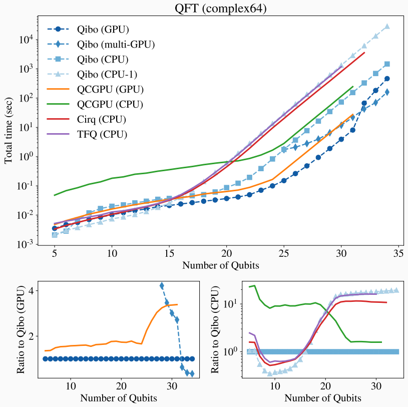


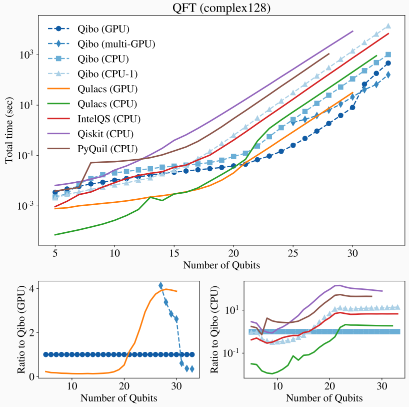
Library	Precision	Hardware
`Qibo 0.1.0`	single double	multi-thread CPU GPU multi-GPU
`Cirq 0.8.1`	single	single-thread CPU
`TFQ 0.3.0`	single	single-thread CPU
`Qiskit 0.14.2`	double	single-thread CPU
`PyQuil 2.20.0`	double	single-thread CPU
`IntelQS 2.0.0`	double	multi-thread CPU
`QCGPU 0.1.1`	single	multi-thread CPU GPU
`Qulacs 0.1.10.1`	double	multi-thread CPU GPU

All computations are performed on the **NVIDIA DGX workstation**.

- GPUs: 4x NVIDIA Tesla V100 with 32GB
- CPU: Intel Xeon E5 with 20 cores with 256 GB of RAM

QFT benchmark



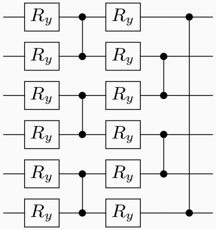


Quantum Fourier Transform simulation performance comparison in single precision (left) and double precision (right).

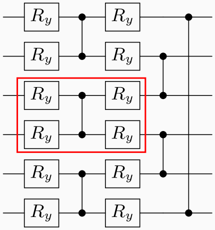
Variational circuit

Variational circuits are inspired by the structure of variational circuits used in **quantum machine learning**.

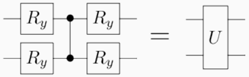
Standard Circuit



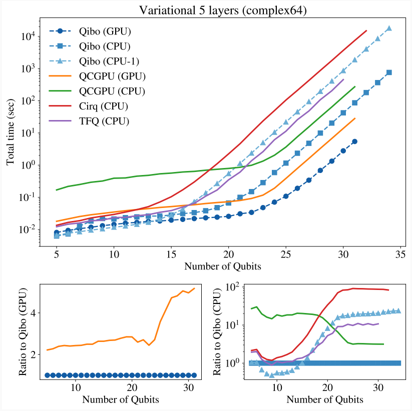
Gate fusion

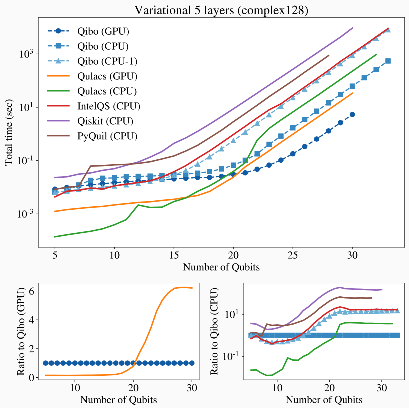


Qibo implements the gate fusion of four R_y and the controlled-phased gate, $C_z \Rightarrow$ applies them as a single two-qubit gate.



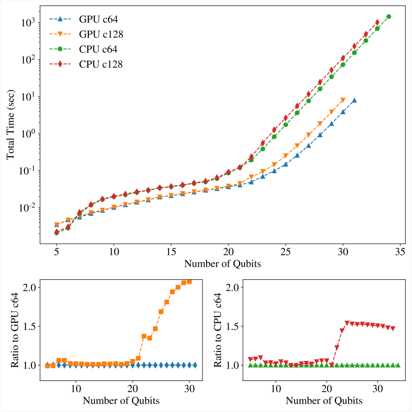
Variational circuit benchmark





Variational circuit simulation performance comparison in single precision (left) and double precision (right).

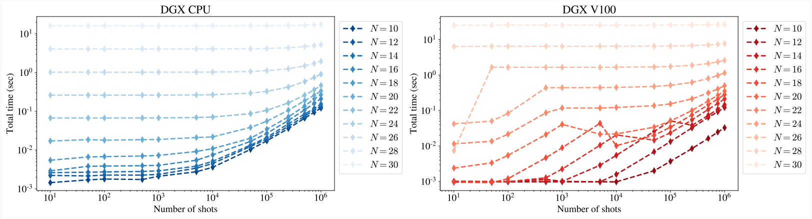
Single vs double precision simulation



Comparison of simulation time when using single (complex64) and double (complex128) precision on GPU and multi- threading (40 threads) CPU.

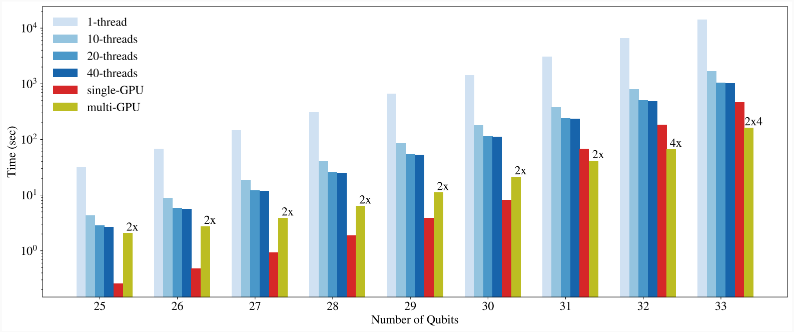
Measurement simulation

Qibo simulates quantum measurements using its standard dense state vector simulator, followed by sampling from the distribution corresponding to the final state vector.



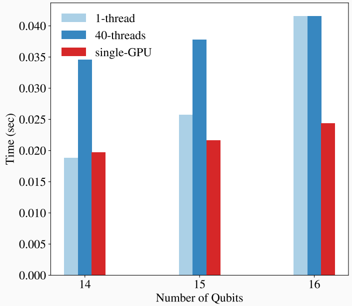
Example of measurement shots simulation on CPU (left) and GPU (right).

Hardware configurations - large circuits

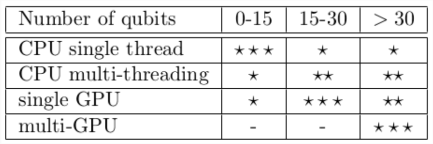


Comparison of Qibo performance for QFT on multiple hardware configurations. For the multi-GPU setup we include a label on top of each histogram bar summarizing the effective number of NVIDIA V100 cards used during the benchmark.

Hardware configurations - small circuits



Comparison of Qibo performance for small QFT circuits on single thread CPU, multi-threading CPU and GPU. Single thread CPU is the optimal choice for up to 15 qubits.



Number of qubits	0-15	15-30	> 30
CPU single thread	$\star\star\star$	$\star$	$\star$
CPU multi-threading	$\star$	$\star\star$	$\star\star$
single GPU	$\star$	$\star\star\star$	$\star\star$
multi-GPU	-	-	$\star\star\star$

Annealing with Qibo

State evolution

Qibo can be used to simulate a unitary time evolution of quantum states.

$$i\partial_t |\psi(t)\rangle = H |\psi(t)\rangle$$

Given an initial state vector $|\psi_0\rangle$ and an evolution Hamiltonian H, the goal is to find the state $|\psi(T)\rangle$ after time T, so that the time-dependent Schrödinger equation:

Adiabatic evolution

Example for adiabatic quantum computation:

Lets consider the evolution Hamiltonian:

$$H(t) = (1 - s(t))H_0 + s(t)H_1,$$

where

- H_0 is a Hamiltonian whose ground state is easy to prepare and is used as the initial condition,
- H_1 is a Hamiltonian whose ground state is hard to prepare
- $s(t)$ is a scheduling function.

According to the adiabatic theorem, for proper choice of $s(t)$ and total evolution time T, the final state $|\psi(T)\rangle$ will approximate the ground state of the “hard” Hamiltonian H_1.

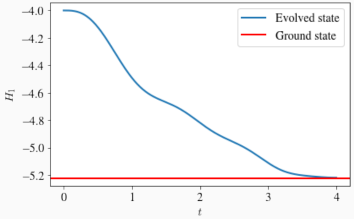
Adiabatic evolution

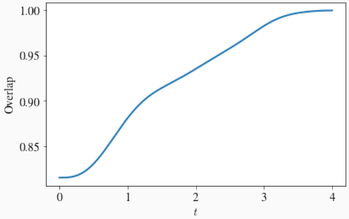
Lets consider the critical transverse field Ising model as the “hard” Hamiltonian:

$$H_0 = -\sum_{i=0}^{N} X_i, \quad H_1 = -\sum_{i=0}^{N} (Z_i Z_{i+1} + X_i)$$

where X_i and Z_i represent the matrices acting on the i-th qubit.

Example with linear scheduler $s(t)$:





Adiabatic evolution

Qibo uses two different methods to simulate time evolution:

- The first method requires constructing the full $2^N \times 2^N$ matrix of H and uses an ordinary differential equation (ODE) solver to calculate the evolution operator $e^{-iH\delta t}$ for a single time step δt and applies it to the state vector via the matrix multiplication

$$|\psi(t + \delta t)\rangle = e^{-iH\delta t} |\psi(t)\rangle$$

Adiabatic evolution

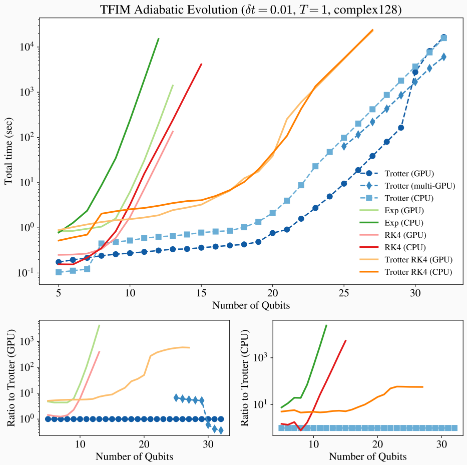
Qibo uses two different methods to simulate time evolution:

- The first method requires constructing the full $2^N \times 2^N$ matrix of H and uses an ordinary differential equation (ODE) solver to calculate the evolution operator $e^{-iH\delta t}$ for a single time step δt and applies it to the state vector via the matrix multiplication

$$|\psi(t+\delta t)\rangle = e^{-iH\delta t} |\psi(t)\rangle$$

- The second time evolution method is based on the Trotter decomposition where local Hamiltonians that contain up to k-body interactions, the evolution operator $e^{-iH\delta t}$ can be decomposed to $2^k \times 2^k$ unitary matrices and therefore time evolution can be mapped to a quantum circuit consisting of k-qubit gates.

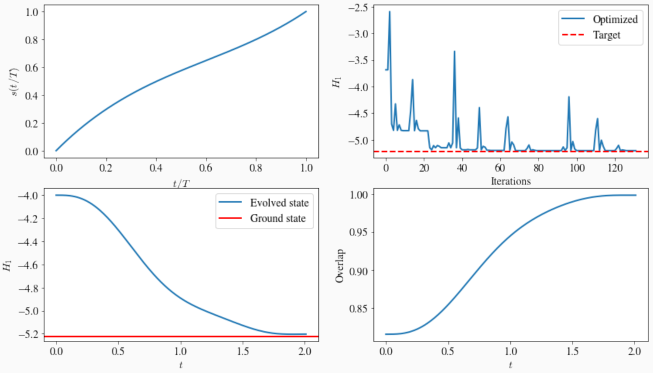
Adiabatic evolution



Adiabatic evolution performance using Qibo and TFIM for extact and Trotter solution.

Scheduling optimization

Example of TFIM scheduling optimization (hybrid algorithm).



Optimization of a polynomial $s(t)$ and final T is performed using classical algorithms while the evolution could be performed by the quantum device.

Applications and tutorials

Qibo applications and tutorial

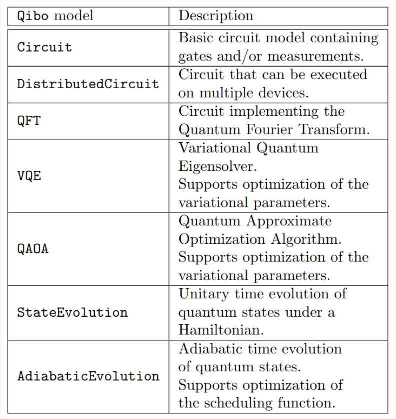
- **Variational circuits**
  - Scaling of variational quantum circuit depth for condensed matter systems
  - Variational Quantum Classifier
  - Data reuploading for a universal quantum classifier
  - Quantum autoencoder for data compression
  - Measuring the tangle of three-qubit states
- **Grover's algorithm**
  - Grover's Algorithm for solving Satisfiability Problems
  - Grover's Algorithm for solving a Toy Sponge Hash function
- **Adiabatic evolution**
  - Simple Adiabatic Evolution Examples
  - Adiabatic evolution for solving an Exact Cover problem
- **Quantum Singular Value Decomposer**
- **Quantum unary approach to option pricing**

See: `https://qibo.readthedocs.io/en/latest/applications.html`

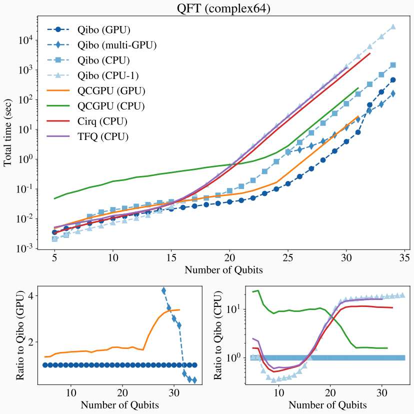
Outlook

Outlook

An efficient hardware accelerated framework for quantum simulation, for the following models:



Qibo model	Description
`Circuit`	Basic circuit model containing gates and/or measurements.
`DistributedCircuit`	Circuit that can be executed on multiple devices.
`QFT`	Circuit implementing the Quantum Fourier Transform.
`VQE`	Variational Quantum Eigensolver. Supports optimization of the variational parameters.
`QAOA`	Quantum Approximate Optimization Algorithm. Supports optimization of the variational parameters.
`StateEvolution`	Unitary time evolution of quantum states under a Hamiltonian.
`AdiabaticEvolution`	Adiabatic time evolution of quantum states. Supports optimization of the scheduling function.



Thank you for your attention.