New SamplerQNN and EstimatorQNN

(changes in qiskit-machine-learning 0.5)

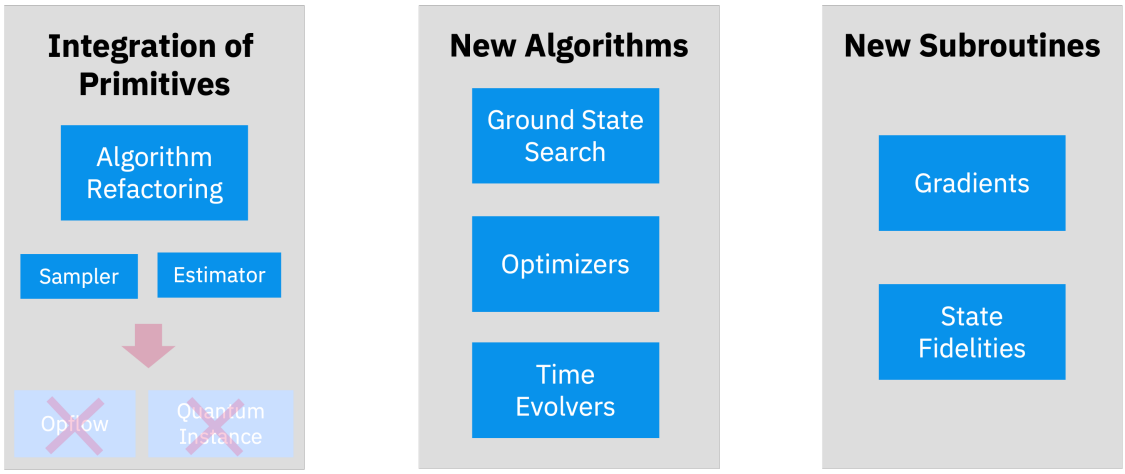
Qiskit Demo Day, Dec. 8 2022

Elena Peña Tapia (ept@zurich.ibm.com)

Index

1. Background

Changes in qiskit.algorithms 0.22



1. Background

New primitive-based gradient framework

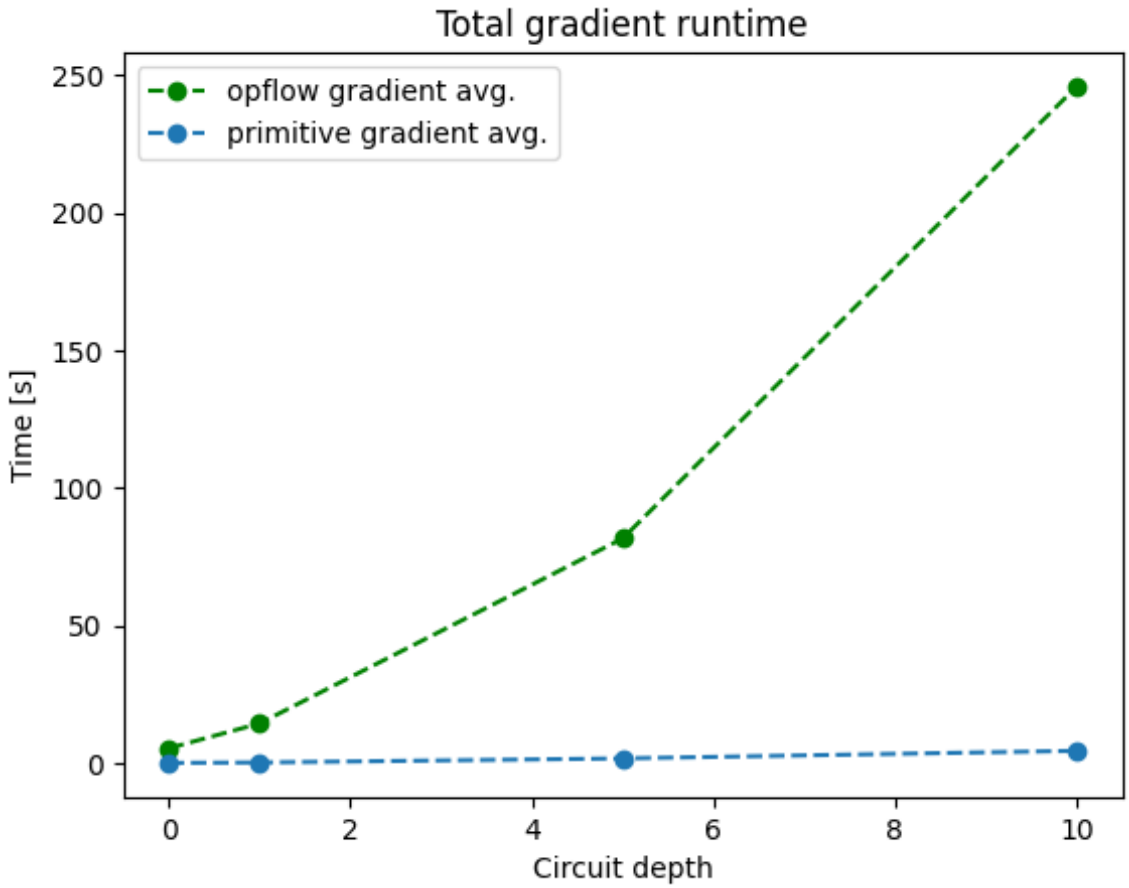
Before: `from qiskit.opflow import Gradient`

Now: `from qiskit.algorithms.gradients import ...`

```
gradient = ParamShiftGrad(estimator)
grad_job = gradient.run([qc], [op], [params])
gradients = grad_job.result().gradients
```

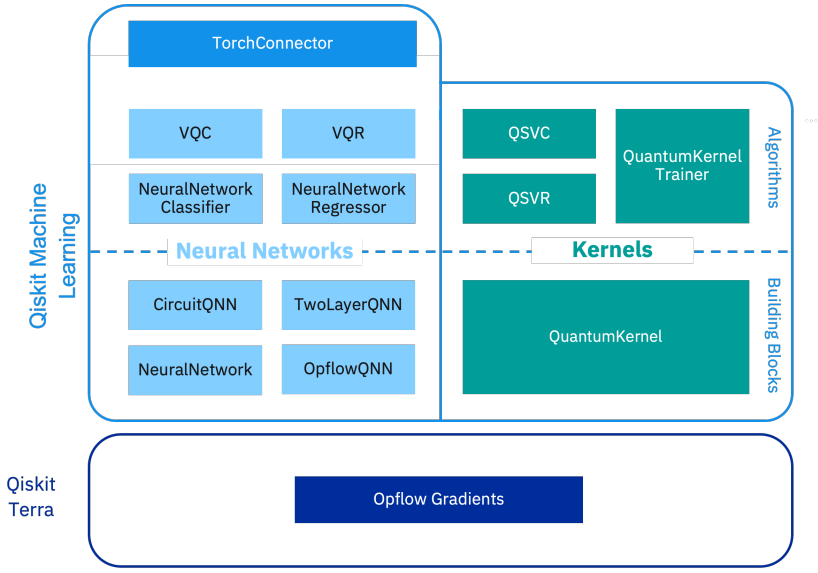
1. Background

New primitive-based gradient framework



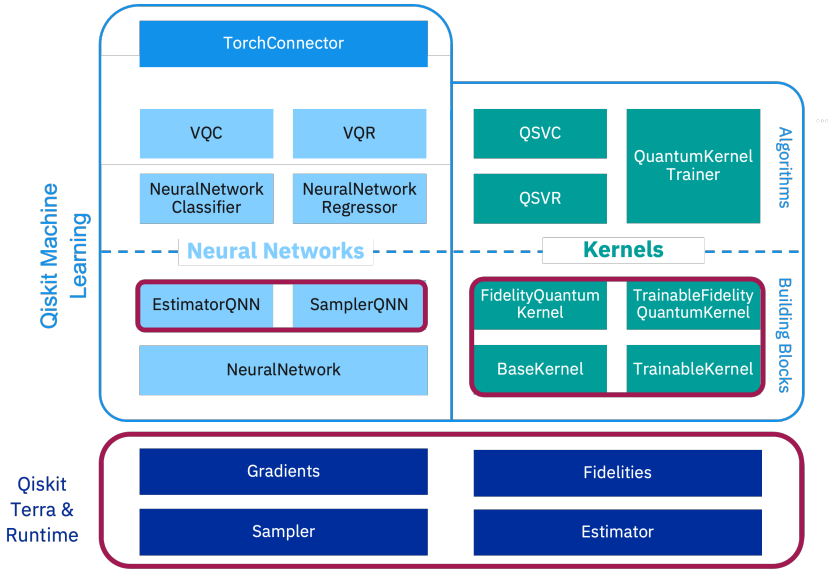
2. Changes in `qiskit-machine-learning`

`0.4` Diagram:



2. Changes in `qiskit-machine-learning`

`0.5` Diagram:



3. Example: `EstimatorQNN` vs. `OpflowQNN`

OpflowQNN

```python from qiskit import IBMQ from qiskit.utils import QuantumInstance from qiskit.neural_networks import OpflowQNN # define provider IBMQ.load_account() provider = IBMQ.get_provider(project="default") device = provider.get_backend("ibm_geneva") # define quantum instance qi = QuantumInstance(backend=device) ```

```python # construct circuit qc = RealAmplitudes(1) qc_sfn = StateFn(qc) # construct observable H1 = StateFn(PauliSumOp.from_list([('Z', 1.0), ('X', 1.0)])) H2 = StateFn(PauliSumOp.from_list([('Y', 1.0)])) # combine observable and circuit to get objective function op1 = ~H1 @ qc_sfn op2 = ListOp([op1, ~H2 @ qc_sfn]) ```

EstimatorQNN

```python from qiskit_ibm_runtime import QiskitRuntimeService, \ Session, Estimator from qiskit.neural_networks import EstimatorQNN # define service service = QiskitRuntimeService(channel="ibm_quantum") # define session with Session(service=service, backend="ibm_geneva") as session: # define estimator estimator = Estimator() ```

```python # construct circuit qc = RealAmplitudes(1) # construct observable obs1 = SparsePauliOp.from_list([('Z', 1.0), ('X', 1.0)]) obs2 = SparsePauliOp.from_list([('Y', 1.0)]) ```

3. Example: `EstimatorQNN` vs. `OpflowQNN`

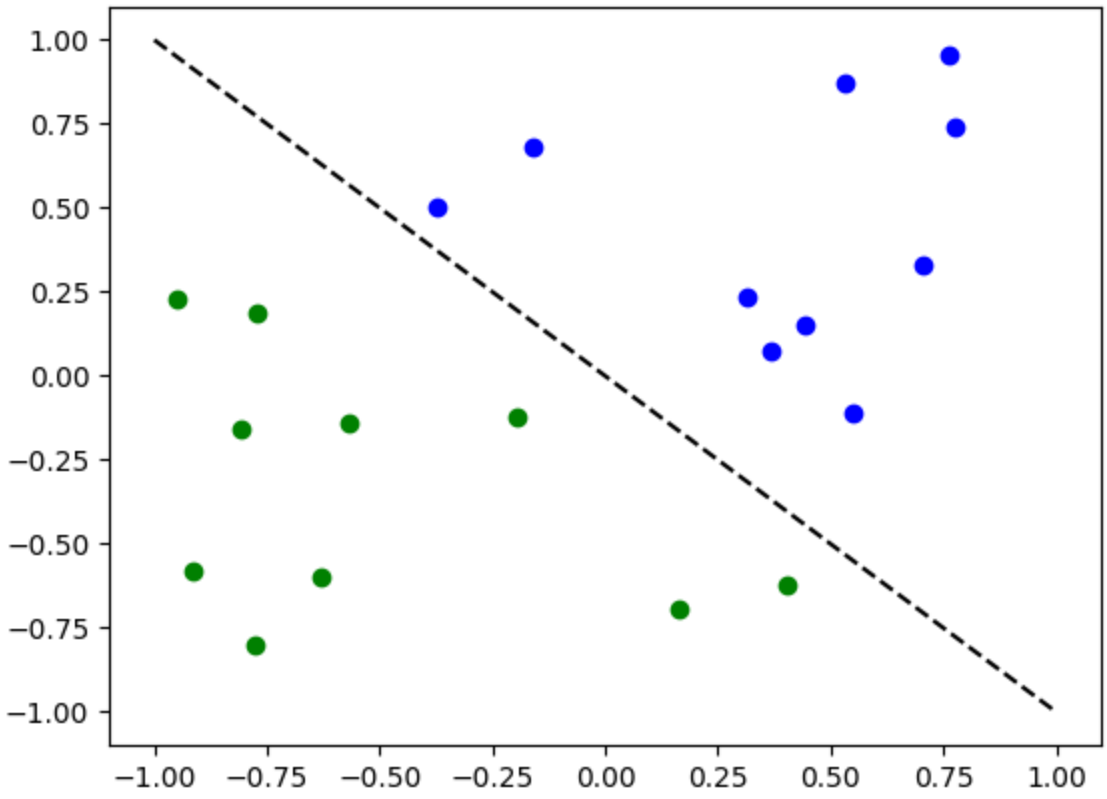
OpflowQNN	EstimatorQNN
``python # define QNN qnn1 = OpflowQNN(operator=op1, input_params= [qc.parameters[:2]], weight_params= [qc.parameters[2:]], quantum_instance=qi) # forward pass fwd = qnn1.forward(input, weights) ```	```python # with Session(service=service, backend="ibm_geneva") as session: # define QNN qnn1 = EstimatorQNN(estimator=estimator, circuit=qc, observables=obs1, input_params= [qc.parameters[:2]], weight_params= [qc.parameters[2:]]) # forward pass fwd = qnn1.forward(input, weights) ```
``python # backward pass bckwd = qnn1.backward(input, weights) # define more complex QNN qnn2 = OpflowQNN(operator=op2, input_params=qc.parameters[0], weight_params=qc.parameters[1], quantum_instance=qi) ```	```python # backward pass bckwd = qnn1.backward(input, weights) # define more complex QNN qnn2 = EstimatorQNN(estimator=estimator, circuit=qc, observables=[obs1, obs2], input_params= [qc.parameters[0]], weight_params= [qc.parameters[1]]) ```

3. Example: Classification with `SamplerQNN`

In [11]:
```
import numpy as np
import matplotlib.pyplot as plt

num_inputs = 2
num_samples = 20
X = 2 * algorithm_globals.random.random([num_samples, num_inputs]) - 1
y01 = 1 * (np.sum(X, axis=1) >= 0)
y = 2 * y01 - 1
y_one_hot = np.zeros((num_samples, 2))
for i in range(num_samples):
    y_one_hot[i, y01[i]] = 1
```

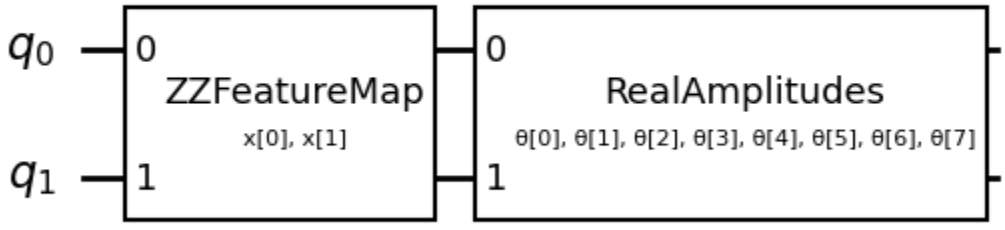
In [12]:
```
# Plot dataset
for x, y_target in zip(X, y):
    if y_target == 1:
        plt.plot(x[0], x[1], "bo")
    else:
        plt.plot(x[0], x[1], "go")
plt.plot([-1, 1], [1, -1], "--", color="black")
```

Out[12]:
```
[<matplotlib.lines.Line2D at 0x7fd7f0d75550>]
```



In [4]:
```
from qiskit import QuantumCircuit
from qiskit.circuit.library import ZZFeatureMap, RealAmplitudes

# construct QNN
qc = QuantumCircuit(2)
feature_map = ZZFeatureMap(2)
ansatz = RealAmplitudes(2)
qc.compose(feature_map, inplace=True)
qc.compose(ansatz, inplace=True)
qc.draw("mpl", style="bw")
```

Out[4]:

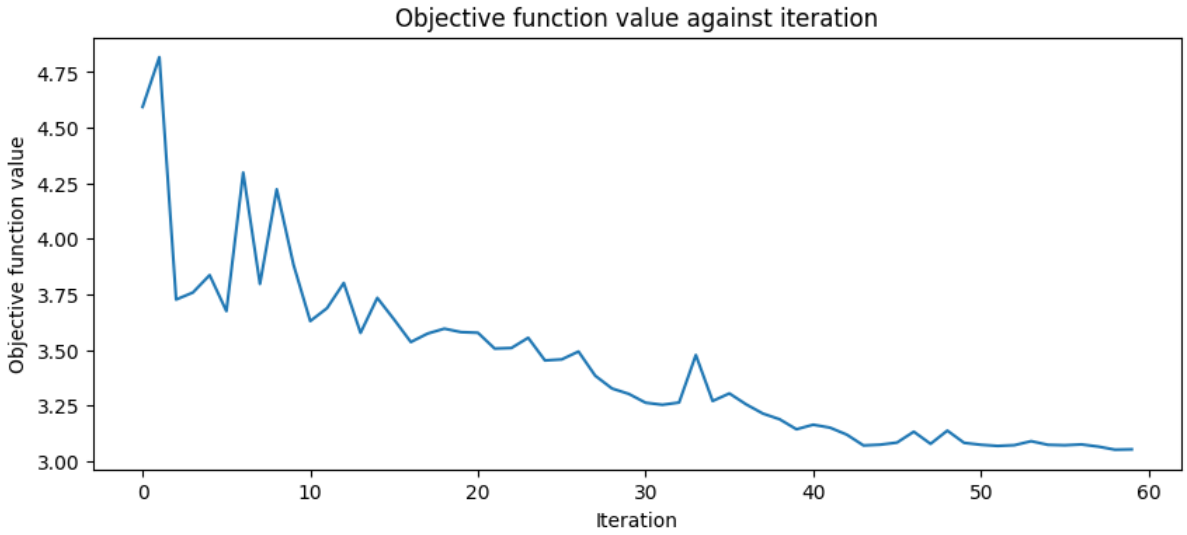


In [8]:
```
from qiskit.primitives import Sampler
from qiskit_machine_learning.neural_networks import SamplerQNN

sampler = Sampler()
qnn1 = SamplerQNN(
    sampler=sampler,
    circuit=qc,
    input_params=feature_map.parameters,
    weight_params=ansatz.parameters,
)
```

In [13]:
```
# callback function that draws a live plot when the .fit() method is called
from IPython.display import clear_output


def callback_graph(weights, obj_func_eval):
    clear_output(wait=True)
    objective_func_vals.append(obj_func_eval)
    plt.title("Objective function value against iteration")
    plt.xlabel("Iteration")
    plt.ylabel("Objective function value")
    plt.plot(range(len(objective_func_vals)), objective_func_vals)
    plt.show()
```

In [14]:
```
from qiskit_machine_learning.algorithms import NeuralNetworkClassifier
from qiskit.algorithms.optimizers import COBYLA

# construct neural network classifier
estimator_classifier = NeuralNetworkClassifier(
    qnn1, optimizer=COBYLA(maxiter=60), callback=callback_graph)
```

In [15]:
```
# create empty array for callback to store evaluations of the objective func
objective_func_vals = []
plt.rcParams["figure.figsize"] = (10, 4)

# fit classifier to data
estimator_classifier.fit(X, y)

# return to default figsize
plt.rcParams["figure.figsize"] = (6, 4)

# score classifier
estimator_classifier.score(X, y)
```



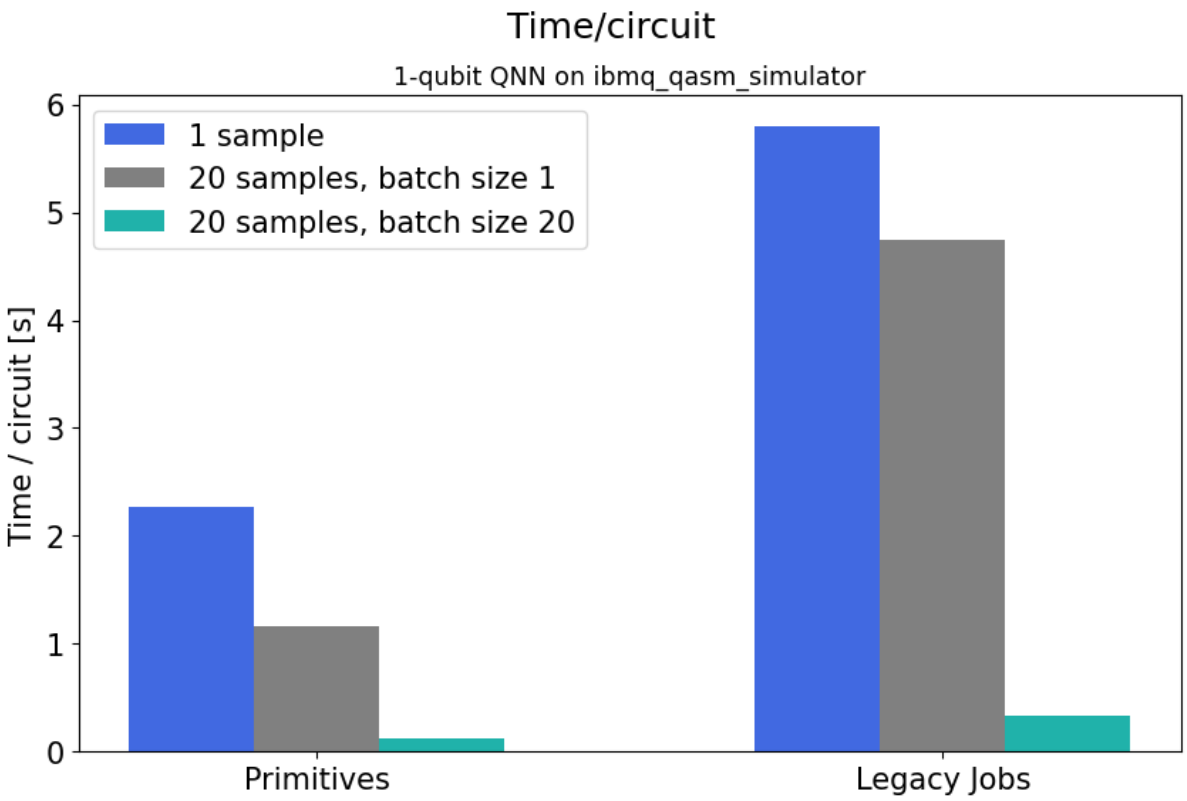
Out[15]: 0.2

In [17]:
```
from matplotlib import pyplot as plt
import numpy as np

plt.rcParams.update({"font.size": 15})

benchmark_data = {}
benchmark_data["Primitives"] = (2.272801638, 1.1564188, 0.125061715)
benchmark_data["Legacy Jobs"] = (5.794027249, 4.741929928, 0.330515218)

X = ["Primitives", "Legacy Jobs"]
X_axis = np.arange(len(X))

qnn_benchmark = plt.figure(figsize=(10, 6))
# plt.yscale("log")
plt.ylabel("Time / circuit [s]")
plt.xticks(X_axis, X)
plt.suptitle("Time/circuit")
plt.title("1-qubit QNN on ibmq_qasm_simulator", fontsize="small")
for x, (key, val) in enumerate(benchmark_data.items()):
    line_1 = plt.bar(x - 0.2, val[0], 0.2, color="royalblue")
    line_2 = plt.bar(x, val[1], 0.2, color="grey")
    line_3 = plt.bar(x + 0.2, val[2], 0.2, color="lightseagreen")

plt.legend(
    handles=[line_1, line_2, line_3],
    labels=["1 sample", "20 samples, batch size 1", "20 samples, batch size
)
plt.close()
```

4. Performance: Preliminary benchmark, SamplerQNN

In [18]:
```
qnn_benchmark
```

Out[18]:



5. Wrap-Up

`qiskit-machine-learning 0.5` uses primitives!!

- New gradient framework useful for QNNs
- New QNN classes: `SamplerQNN` and `EstimatorQNN`
- Easier design with custom observables
- Promising performance improvements

For more info, check out the QNNs tutorial on GitHub!

In [1]:
```
import qiskit.tools.jupyter

%qiskit_version_table
%qiskit_copyright
```

Version Information

Qiskit Software	Version
qiskit-terra	0.23.0.dev0+3ce1737
qiskit-aer	0.11.0
qiskit-ibmq-provider	0.19.2
qiskit	0.39.0
qiskit-nature	0.5.0
qiskit-machine-learning	0.5.0
System information	
Python version	3.9.13
Python compiler	Clang 12.0.0
Python build	main, Oct 13 2022 16:12:30
OS	Darwin
CPUs	8
Memory (Gb)	64.0
	Mon Nov 07 18:00:23 2022 CET

This code is a part of Qiskit