

Introduction to Using Samplers and Estimators in Quantum Computing

Quantum computers operate fundamentally differently from classical computers. Classical computers process information deterministically, meaning they will always produce the same output given the same input. Quantum computers, on the other hand, operate probabilistically. This means that the outcome of a quantum computation can vary, even with the same input, due to the inherent probabilistic nature of quantum mechanics.

Why Use Samplers and Estimators?

Samplers and **Estimators** are critical for running quantum algorithms efficiently:

1. Samplers:

- **Purpose:** Used to gather and analyze the distribution of possible outcomes from a quantum circuit. They help understand the probabilistic output by sampling multiple circuit runs.
- **Use Case:** For tasks like generating samples according to the quantum state prepared by a circuit (e.g., the relevant states/counts of a quantum circuit). This is useful in various quantum algorithms and for verifying quantum states.

2. Estimators:

- **Purpose:** Used to estimate expected values of certain observables (e.g., energy in a quantum system) based on the probabilistic outcomes.
- **Use Case:** Essential for algorithms in quantum chemistry and physics where measuring specific properties of a quantum system is required.

The transition from `backend.run()` to Samplers and Estimators

With the deprecation of *backend.run()* in Qiskit, there is a shift towards using Samplers and Estimators for executing quantum circuits. The *backend.run()* method and related session methods are deprecated as of Qiskit-IBM-Runtime 0.23 and will be removed in future releases. This change encourages the use of the more robust and [flexible primitives](#): “*sampler.run()*” and “*estimator.run()*”

Methods and Objects in Quantum Computing

To better understand samplers and estimators, let's briefly explain some important methods and objects used in quantum computing when using Samplers and Estimators:

- **Isa_circuit:** A term used to describe a circuit mapped onto a given quantum processor's specific hardware topology and constraints. This involves layout adjustments and optimizations to ensure efficient execution on the target backend.
- **Pass_manager:** A tool that manages the optimization passes applied to a quantum circuit. These passes optimize the circuit in various ways, such as reducing gate count or adapting the circuit to the hardware constraints.
- **Transpile:** A function that converts a high-level quantum circuit into a form that can be executed on a specific quantum backend. This process includes mapping logical qubits to physical qubits, optimizing the circuit, and ensuring it adheres to the backend's constraints.
- **Observable:** An operator representing a physical quantity to be measured in a quantum system. In the context of estimators, observables are often represented by Pauli operators (e.g., Z, X, Y) applied to qubits.
- **Counts:** The number of times each possible outcome occurs when a quantum circuit is executed multiple times. This is typically represented as a dictionary where keys are bitstrings (representing the states) and values are the counts of those bitstrings.

- **Expectation values:** The weighted average of the measurement outcomes of an observable, providing a single number that represents the average result of measuring that observable on a quantum state. Expectation values are crucial for algorithms in quantum chemistry, optimization, and machine learning.

How to Use Samplers and Estimators

Using Samplers and Estimators involves several steps in the Qiskit framework, IBM's open-source quantum computing software development kit.

1. Setup:

- Make sure you have Qiskit and Qiskit-IBM-Runtime installed

```
[1] !pip install qiskit
    !pip install qiskit-ibm-runtime
```

2. Initialization:

- Import the necessary classes from Qiskit-IBM-Runtime.

```
[18] from qiskit_ibm_runtime import SamplerV2 as Sampler, EstimatorV2 as Estimator
```

- Initializing Qiskit IBM Quantum Runtime

```
from qiskit_ibm_runtime import QiskitRuntimeService

# If you did not previously save your credentials, use the following line instead:
service = QiskitRuntimeService(channel="ibm_quantum", token="<Your IBM API Token>")
```

3. Using Samplers:

- Import the necessary libraries and generate a Quantum Circuit

```

✓ 0s [7] import numpy as np
      from qiskit.circuit.library import IQP
      from qiskit.transpiler.preset_passmanagers import generate_preset_pass_manager
      from qiskit.quantum_info import random_hermitian

      n_qubits = 127

      mat = np.real(random_hermitian(n_qubits, seed=1234))
      circuit = IQP(mat)
      circuit.measure_all()

```

- Use the pass manager class to create an ISA circuit for the Quantum Hardware

```

✓ 45s [8] pm = generate_preset_pass_manager(optimization_level=1, backend=backend)
      isa_circuit = pm.run(circuit)

```

- Initialize the Sampler

```

✓ 0s [9] sampler = Sampler(mode=backend)

```

- Using the Sampler, run the circuit

```

[10] job = sampler.run([isa_circuit])
      print(f">>> Job ID: {job.job_id()}")
      print(f">>> Job Status: {job.status()}")

>>> Job ID: csx1yc97ynng008zexwg
>>> Job Status: QUEUED

```

- Get the job results and print out the counts

```

✓ 0s [9] result = job2.result()

      # Get results for the first (and only) PUB
      pub_result = result[0]
      print(f"Counts for the meas output register: {pub_result.data.meas.get_counts()}")

Counts for the meas output register: {'10101010100001101001001000010100000101100001

```

4. Using Estimators:

- ```
import numpy as np
from qiskit.circuit.library import IQP
from qiskit.transpiler.preset_passmanagers import generate_preset_pass_manager
from qiskit.quantum_info import SparsePauliOp, random_hermitian

n_qubits = 50

mat = np.real(random_hermitian(n_qubits, seed=1234))
circuit = IQP(mat)
observable = SparsePauliOp("Z" * n_qubits)
print(f">>> Observable: {observable.paulis}")
```

- ```
[4] pm = generate_preset_pass_manager(optimization_level=1, backend=backend)
isa_circuit = pm.run(circuit)
isa_observable = observable.apply_layout(isa_circuit.layout)
```

- ```
[5] estimator = Estimator(mode=backend)
```

- ```
job = estimator.run([(isa_circuit, isa_observable)])
print(f">>> Job ID: {job.job_id()}")
print(f">>> Job Status: {job.status()}")
```
- ```
>>> Job ID: csx1wfssgar0008cyxyg
>>> Job Status: QUEUED
```

- Get the job results and print out the expectation values

```
[] result = job.result()
 print(f">>> {result}")
 print(f" > Expectation value: {result[0].data.evs}")
 print(f" > Metadata: {result[0].metadata}")

➡ >>> PrimitiveResult([PubResult(data=DataBin(evs=np.ndarray(
 > Expectation value: 0.4482758620689655
 > Metadata: {'shots': 4096, 'target_precision': 0.015625,
```

## 5. Advanced Features:

- The new versions (V2) of these primitives offer enhanced functionality, like efficiently handling vectorized inputs and supporting multiple parameter sets and observables. This flexibility allows for more complex and varied quantum experiments.

Samplers and Estimators are fundamental tools in quantum computing for obtaining measurement outcomes and expectation values. They provide the necessary interface to execute and analyze quantum circuits efficiently, making them indispensable for research and practical applications in quantum computing. Understanding the probabilistic nature of quantum computing and using appropriate primitives like samplers and estimators are crucial for effective quantum computations.

The examples provided illustrate the use of these primitives and highlight the necessary methods and objects involved. By leveraging these tools, one can gather valuable insights from quantum computations, thereby making the most of the unique capabilities of quantum computers.

For more detailed information and advanced usage, refer to the [Migration Guide](#) and IBM Quantum Documentation for [Getting Started with Primitives](#)