Optimizing Qiskit Runtime Primitives for Noise-Resilient Quantum Computing

Mariana Bernagozzi
IBM Quantum

Paula Tristán
IBM Quantum

Primitives programming model

Optimizing Qiskit Runtime Primitives for Noise-Resilient Quantum Computing

MOTIVATION

- +number of qubits, +complexity of workflows
- Not practical to work with individual qubits
- Need for more abstraction layers
- Qiskit proposes two primitives: Sampler and Estimator

QISKIT PRIMITIVES

Sampler

Inputs:
- Circuits (optionally parametrized)

Outputs:
- Sampling quantum states

Estimator

Inputs:
- Circuits (optionally parametrized)
- Observables

Outputs:
- Expectation values

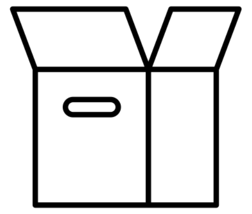
Qiskit SDK
vs.
Qiskit Runtime

QISKIT

QISKIT SDK

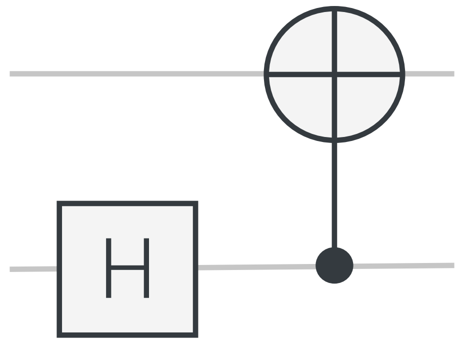
Open-source package

Qiskit SDK is an open-source Python package (package name: `qiskit`).



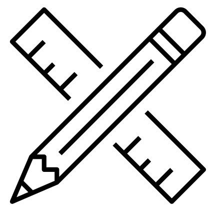
Manage circuits and operators

The Qiskit SDK facilitates working with quantum computers at the level of quantum circuits, operators, and primitives.



Primitives interface

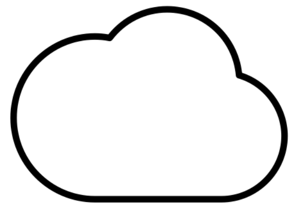
Qiskit SDK defines the interface for the primitives. It also provides reference implementation, from which different quantum hardware providers can derive their own.



QISKIT RUNTIME

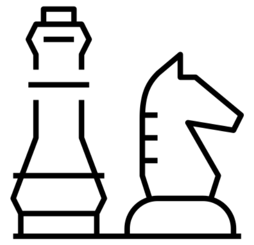
Cloud-based service

Qiskit Runtime is a cloud-based service for executing quantum computations on IBM hardware.



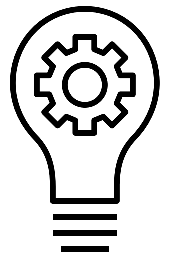
Fast classical-quantum executions

Qiskit Runtime is strategically located near Quantum computing units (QPUs) allowing for low-latency classical-quantum executions.



Primitives implementation

Qiskit Runtime provides an efficient implementation of the primitives, leveraging techniques such as error suppression and error mitigation.



Optimizing Qiskit Runtime Primitives for Noise-Resilient Quantum Computing

Error suppression and error mitigation embedded in Qiskit Runtime Primitives

Optimizing Qiskit Runtime Primitives for Noise-Resilient Quantum Computing

MOTIVATION	TECHNIQUES: Error suppression	TECHNIQUES: Error mitigation
• Quantum executions are susceptible to noise and errors. • Very active area of research. • Leverage low-latency between classical and quantum world to implement these techniques.	• Low quantum computational overhead • Aims to minimize the occurrence of errors in the first place rather than detecting and correcting them after the execution. • Modifies the input circuits in a targeted way • Examples: dynamical decoupling, twirling, and gate optimization.	• Introduces additional quantum computational overhead • Allows errors to occur and then infers better results from multiple noisy calculations. • Classical post-processing is typically used to combine the outputs and infer better results • Examples: TREX, ZNE, PEC, PEA.

The long journey to Primitives V2

Backend.run	Primitives V1	Primitives V2
• No encapsulation of common behavior. • Too low level for developing applications. • No error mitigation.	• Introduction of Sampler and Estimator. • Provide an abstraction layer. • Embedded error mitigation (non-flexible options). • Non-scalable transformation of abstract circuits to ISA circuits.	• Introduction of Primitives Unified Blocs (PUBs). • More configurable error mitigation options. • Transformation to ISA circuits implemented as a service, outside the Primitives.

Primitives V2: Primitive Unified Blocs (PUBs) and broadcasting rules

A PUB is a *single circuit* along with *auxiliary data* required to execute the circuit relative to the primitive in question.

EstimatorPub

1. Single circuit
2. An `ObervablesArray`
3. A `BindingsArray`

SamplerPub

1. A single circuit
2. A `BindingsArray`

Interface change drastically

Estimator V1

List-based interface:

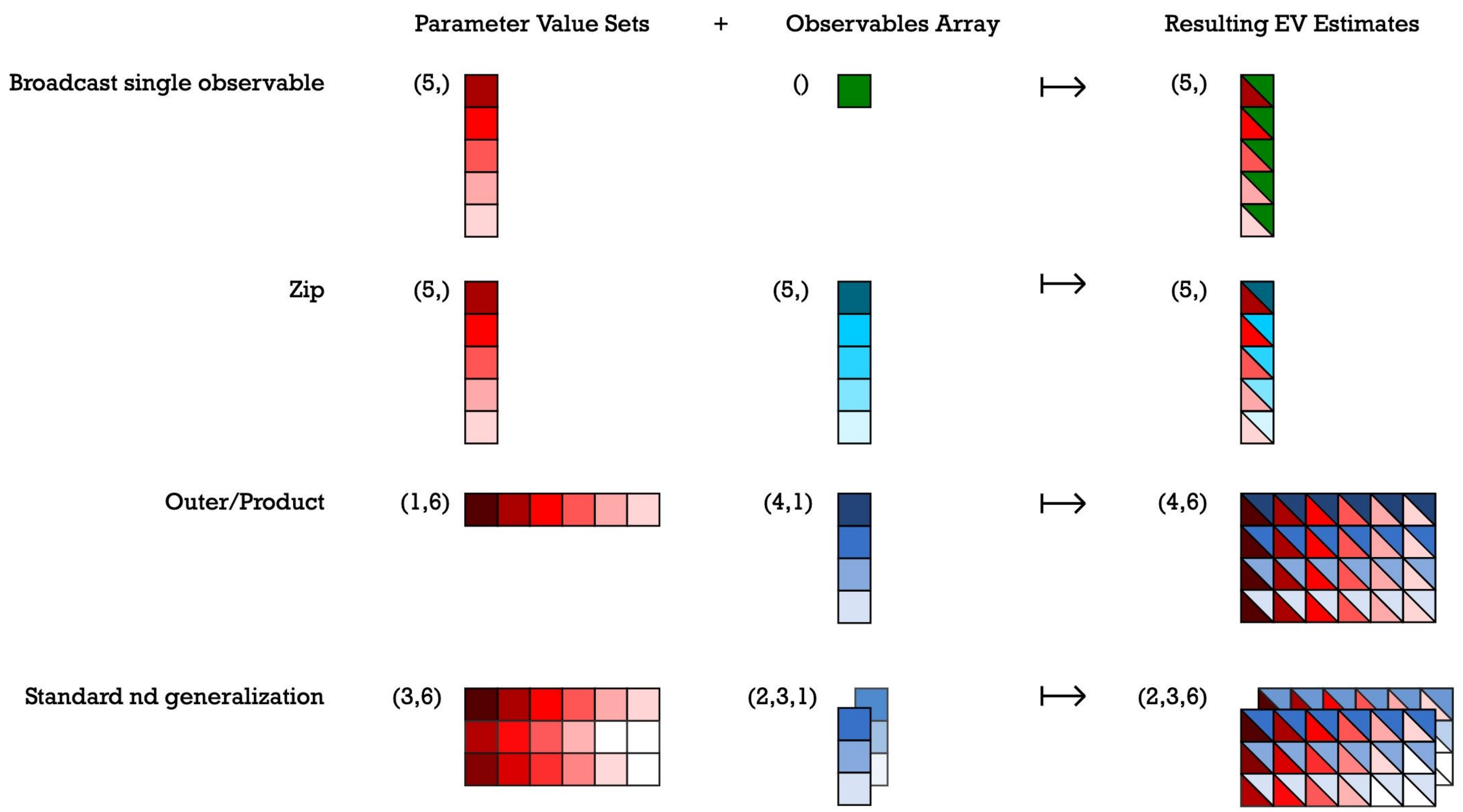
```
Estimator.run(
  [circuit1, circuit2, circuit3],
  [obs1,     obs2,     obs3    ],
  [params1,  params2,  params3 ]
 )
```

Estimator V2

PUB-based interface:

```
Estimator.run(
  (circuit1, [obs1, obs2], [params1],
  (circuit2, [obs3      ], [params2]),
  (circuit3, [obs1, obs2], [ps1][ps2])
 )
```

Broadcasting rules in a PUB



Primitives V2: Configurable error suppression and error mitigation

Dynamical Decoupling options

Options for dynamical decoupling, such as the *sequence* and *scheduling method* to use.

Twirling options

Twirling options, such as whether to apply twirling to *gates* and/or *measurements*, and the *number of shots* to run for each random sample.

Resilience options

Advanced options for configuring error mitigation methods such as ZNE, PEC, or PEA.

Execution options

Including whether to *initialize qubits* and the *repetition delay*.

Key Takeaways

1

The computational model introduced by Qiskit consists of two primitives: *Sampler* and *Estimator*.

2

Primitive interfaces are defined in Qiskit SDK and there can be many implementations. The ones managed by IBM are called *Qiskit Runtime Primitives*.

3

The *Qiskit Runtime Primitives* have evolved over time and will continue to do so.

4

Embedded error mitigation in *Qiskit Runtime Primitives* benefits from low-latency between classical world and QPUs.

5

PUB-based interface is more adequate for majority of workflows.

6

Primitives V2 allows for more customizable error suppression and error mitigation.

Thank you!

Thank you for attending our presentation!

Feedback

Have you used *Qiskit Runtime Primitives* before? We’d love to hear your feedback.

Questions?

Feel welcome to pose your question now or reach out later.