QGSS, 2025

Practical quantum techniques

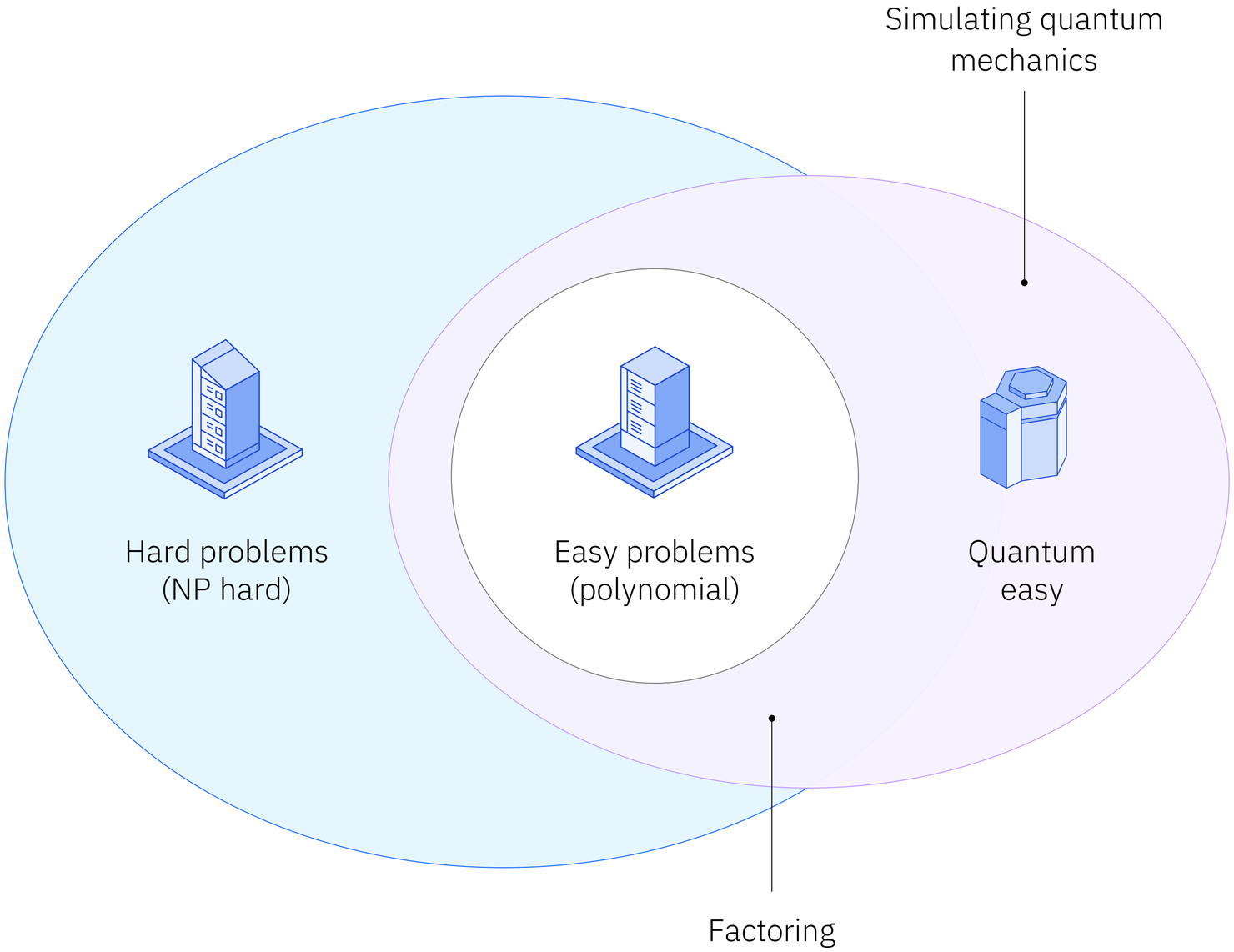
Joana Fraxanet
Quantum Algorithm
Engineering team

IBM **Quantum**

Motivation for scaling quantum algorithms

Quantum computing is not just a faster or better version of classical systems—it is an entirely new branch of computing.

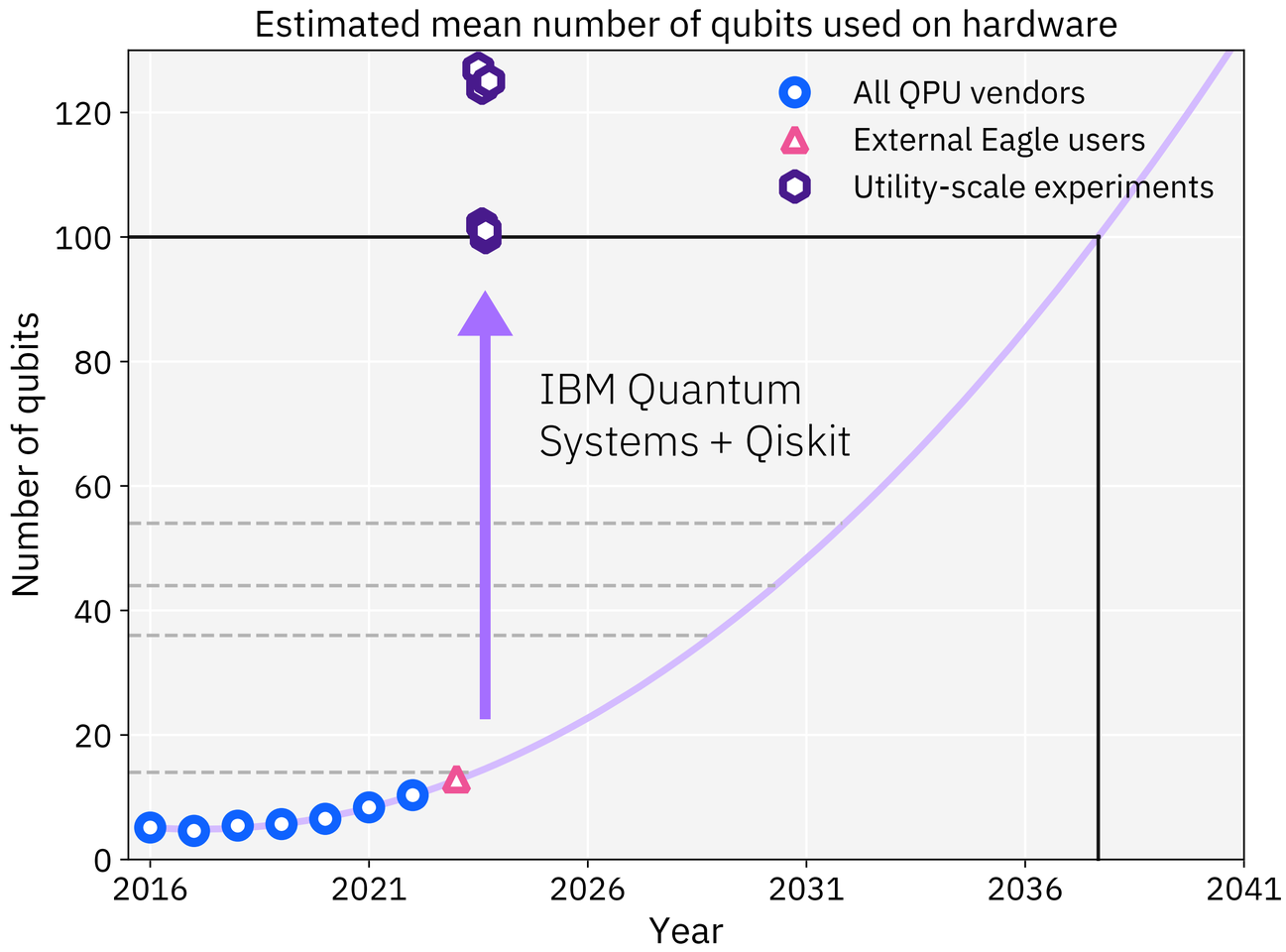
Quantum computers might be able to solve to problems that cannot be tackled classically.



Motivation for scaling quantum algorithms

We need to move beyond simple experiments that can be simulated classically to demonstrate the utility of quantum computing in multiple domains.

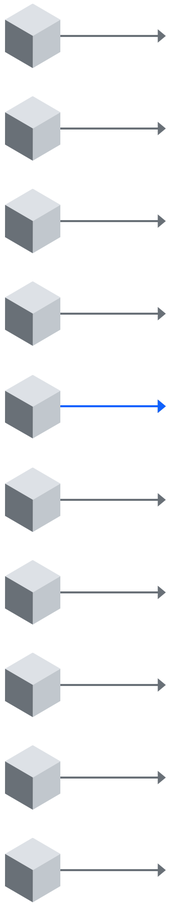
Quantum computers dominate in the high qubit-number regime.

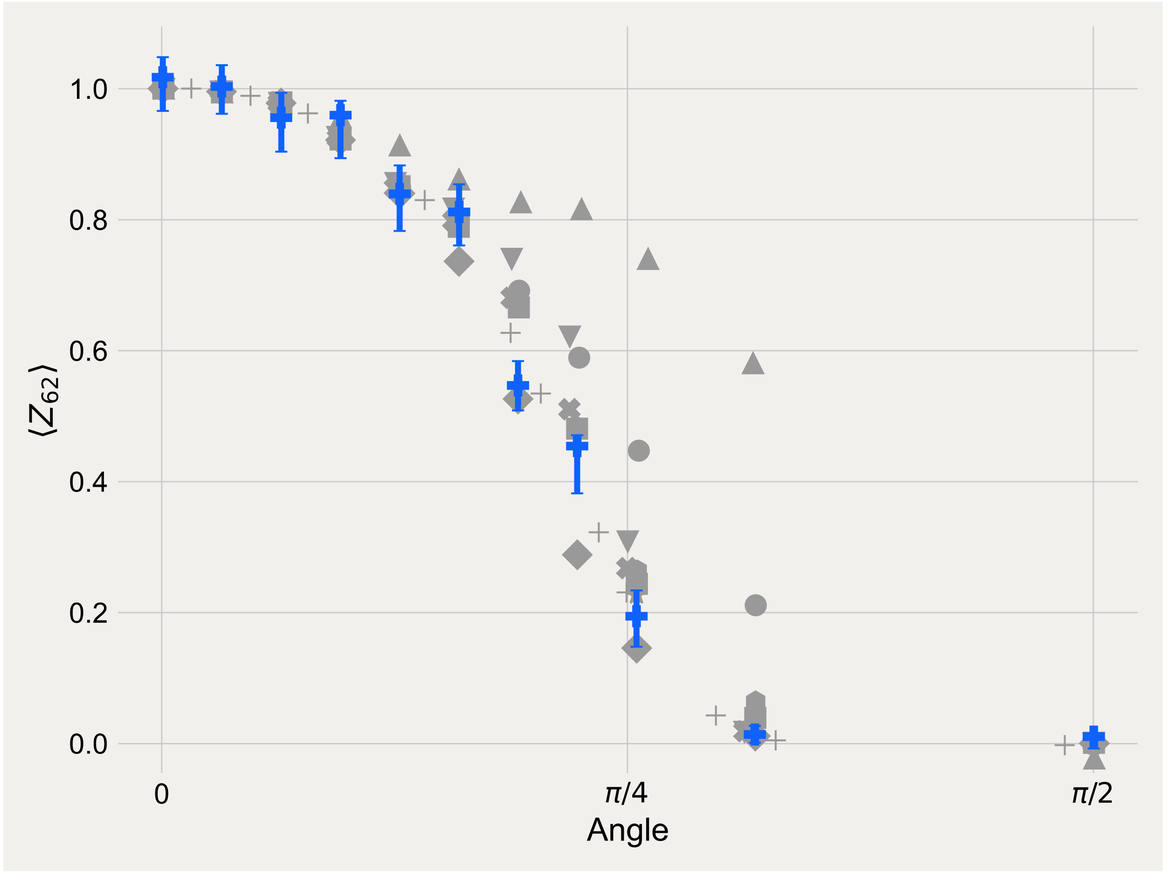


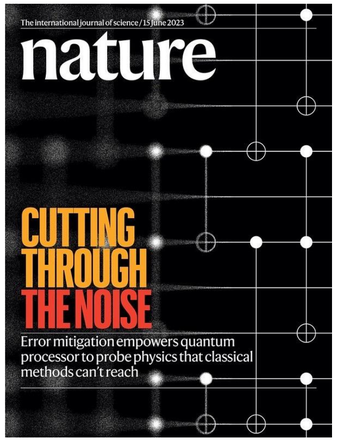
Data for all vendors taken from: arXiv:2307.16130

Motivation for scaling quantum algorithms

IBM Eagle processor (+)





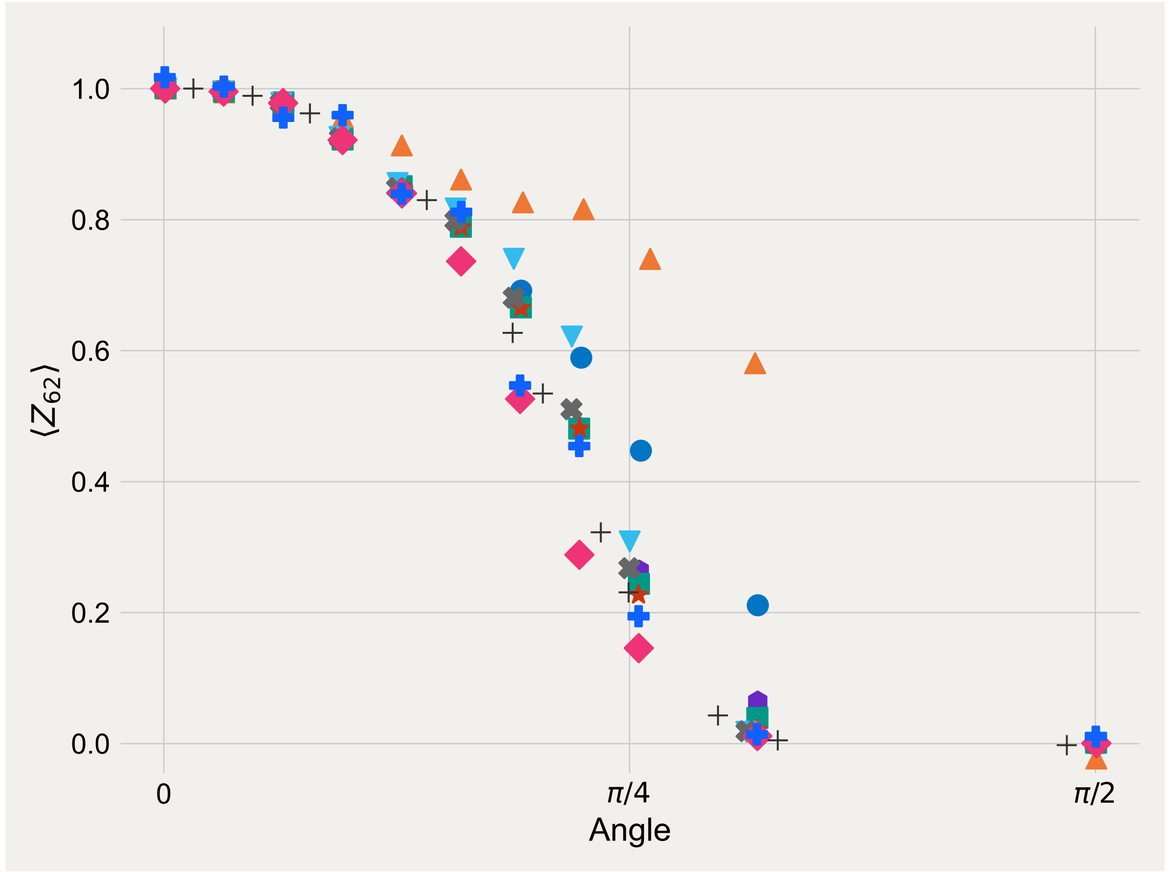


Adapted from PRX Quantum 5, 010308 (2024)

© 2025 IBM Corporation

Motivation for scaling quantum algorithms

MPS 1024 (●)
isoTNS 12 (▲)
TNS (BP) 200 (⬢)
TNS (BP) 500 (★)
IBM Eagle processor (+)
TNS (BP) ∞ (■)
TNO SU ∞ (▼)
Mixed TNS-TNO (✖)
MP0 500 (◆)
31-qubit sim. (+)



Adapted from PRX Quantum 5, 010308 (2024)

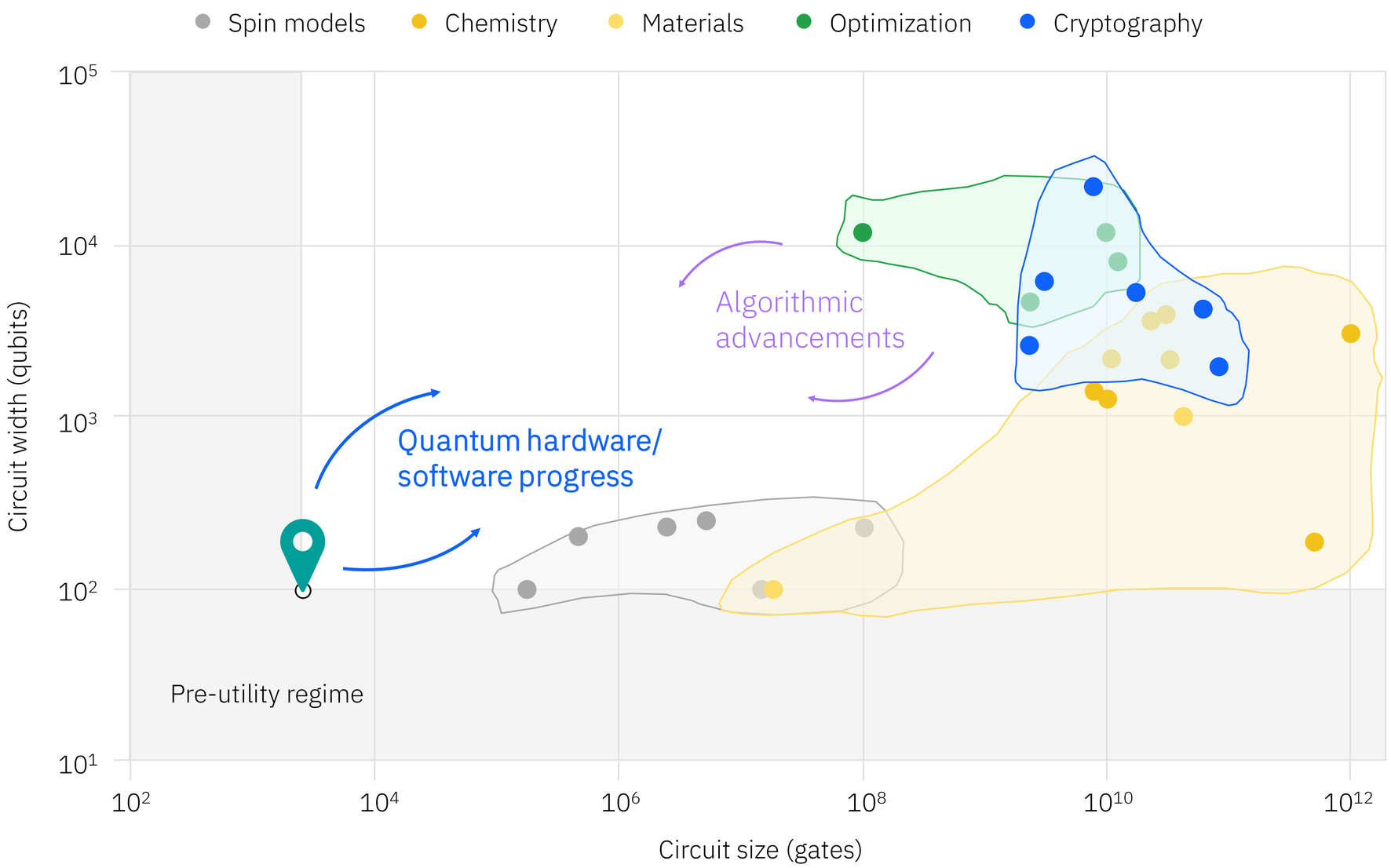
Goal: scaling to regimes where quantum hardware demonstrates measurement of accurate expectation values for circuit volumes at a scale beyond brute-force classical computation.

© 2025 IBM Corporation

Motivation for scaling quantum algorithms

Investments in algorithm research should enable known applications to be realized sooner.

Advances in quantum hardware and software enable new applications or use-cases to be discovered, potentially with current or near-future systems.



Running algorithms on quantum hardware

Qiskit patterns & add-ons

Collection of interoperable research capabilities for enabling algorithm discovery at the utility scale and towards quantum advantage

1. Map

AQC-Tensor

MPF

Qiskit Circuit Library

Input:
Domain inputs

Output:
Circuits, observable

2. Optimize

OBP

Circuit cutting

Transpiler

Input:
Circuits, observable

Output:
ISA circuit, observable

3. Execute

ES and EM

Primitives

Input:
ISA circuit, observable

Output:
Expectation value/samples

4. Post-Process

SQD

M3

Quantum info/visualization

Input:
Expectation value/samples

Output:
Data objects/visualizations

Running algorithms on quantum hardware

AQC-Tensor

MPF

Qiskit Circuit Library

Input:
Domain inputs

Output:
Circuits, observable

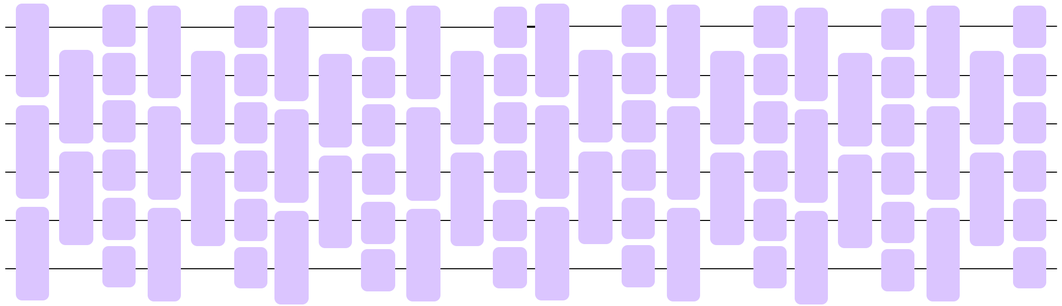
---

1. Map

qiskit.circuit.library

- Standard gates
- Standard directives
- Standard operations
- Generalized gates
- Arithmetic operations
- Basis changes
- Boolean logic
- Data encoding
- Data preparation
- Particular operations
- N-local circuits
- Oracles
- Template circuits

Translate a problem into a quantum circuit and a set of measurements that can reasonably run on quantum hardware.



Example: trotter circuit for a given Hamiltonian.

Let's see how qiskit add-ons allow us to reduce the depth of the circuit .

Running algorithms on quantum hardware

AQC-Tensor

MPF

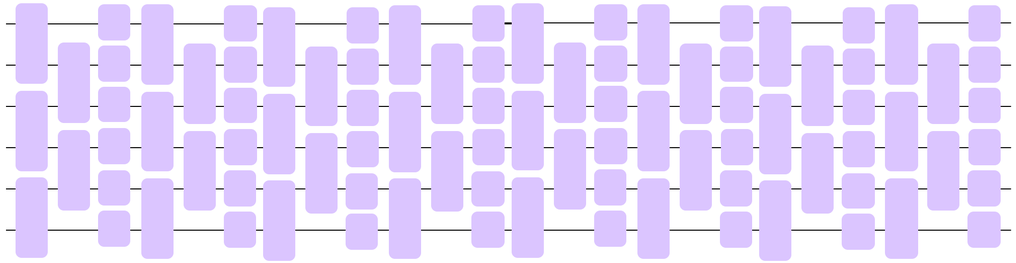
Qiskit Circuit Library

Input:
Domain inputs

Output:
Circuits, observable

---

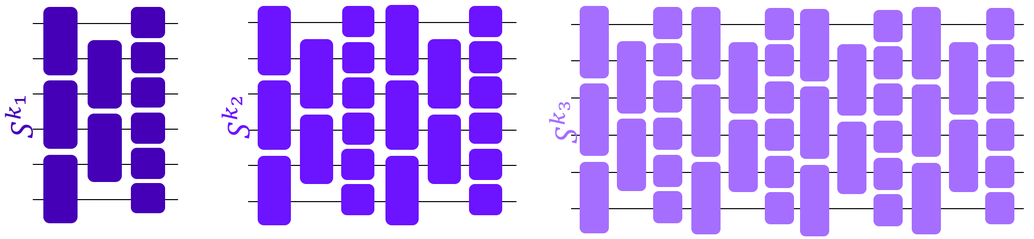
1. Map



Product Formula (PF)

$$S^8\left(\frac{t}{8}\right)$$

Shorter depth + more circuit executions using MultiProduct Formulas (MPF):



$$M(t) = \sum_{j=1}^{l} a_j \, S^{k_j}\left(\frac{t}{k_j}\right)$$

1. Choose number of Trotter steps $k_j = [1,2,4 \ldots]$ and time t
2. Compute coefficients $a_j = [\ldots]$ solving linear system of equations

https://qiskit.github.io/qiskit-addon-mpf/

Running algorithms on quantum hardware

AQC-Tensor

MPF

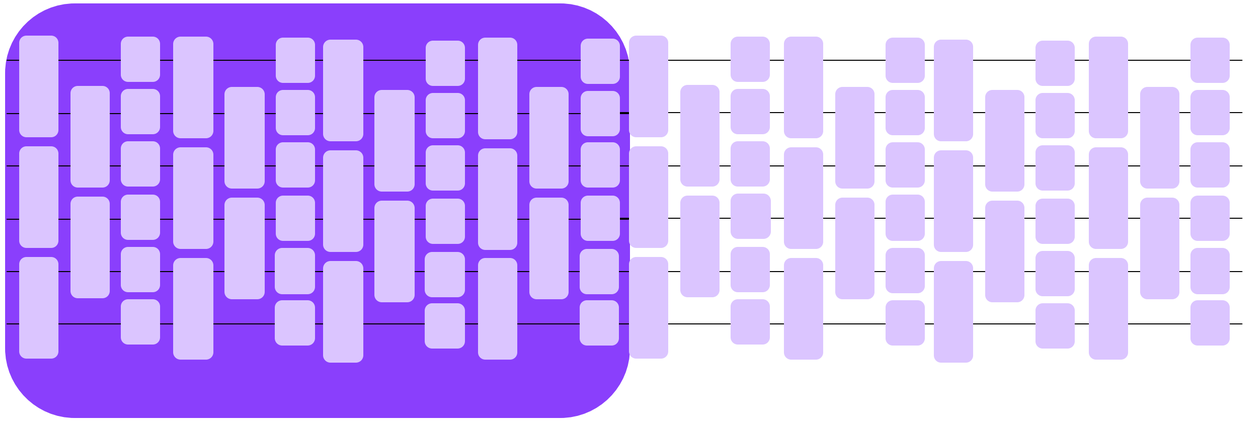
Qiskit Circuit Library

Input:
Domain inputs

Output:
Circuits, observable

---

1. Map



Compress using AQC
Trotterization is applied classically using TEBD and MPS to compress this part of the circuit.

https://qiskit.github.io/qiskit-addon-aqc-tensor/

Running algorithms on quantum hardware

OBP

Circuit cutting

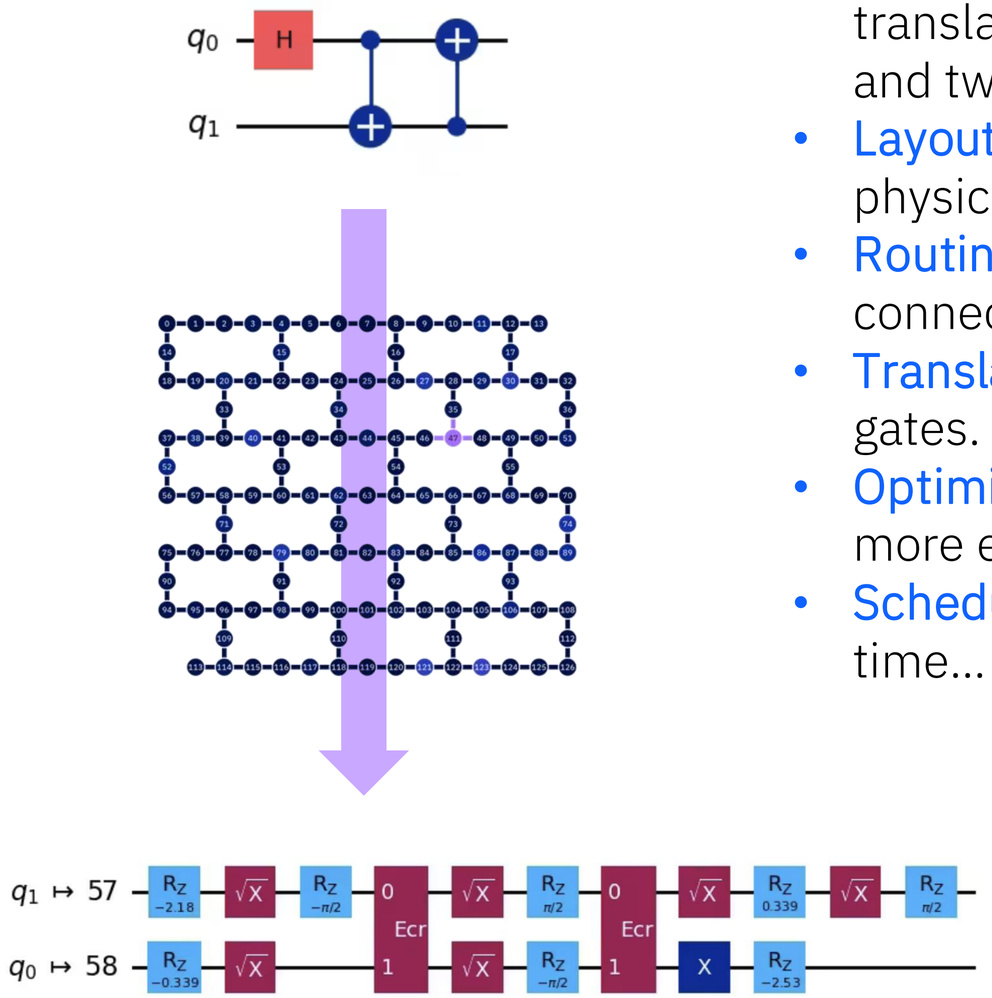
Transpiler

Input:
Circuits, observable

Output:
ISA circuit, observable

---

2. Optimize



- **Init:** Validates the circuit instructions and translates multi-qubit gates into single- and two-qubit gates.
- **Layout:** Mapping the virtual qubits to the physical qubits.
- **Routing:** Inject SWAP gates for connectivity compatibility.
- **Translation:** Translates to QPU's native gates.
- **Optimization:** Optimization loop to find more efficient decompositions.
- **Scheduling:** Scheduling method, idle time…

Running algorithms on quantum hardware

OBP

Circuit cutting

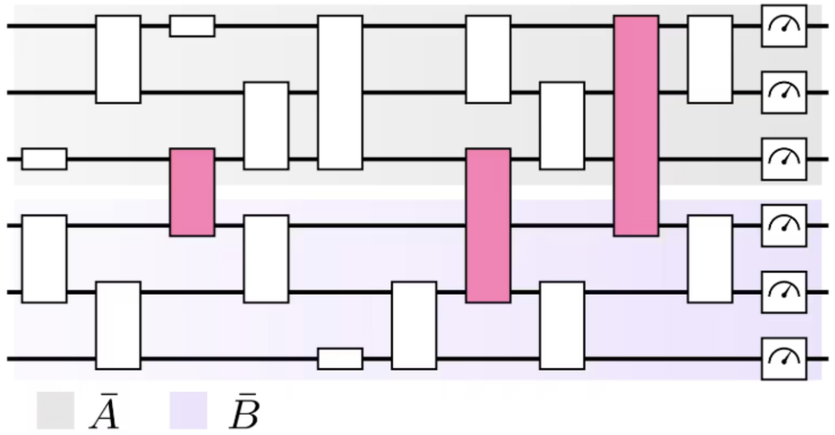
Transpiler

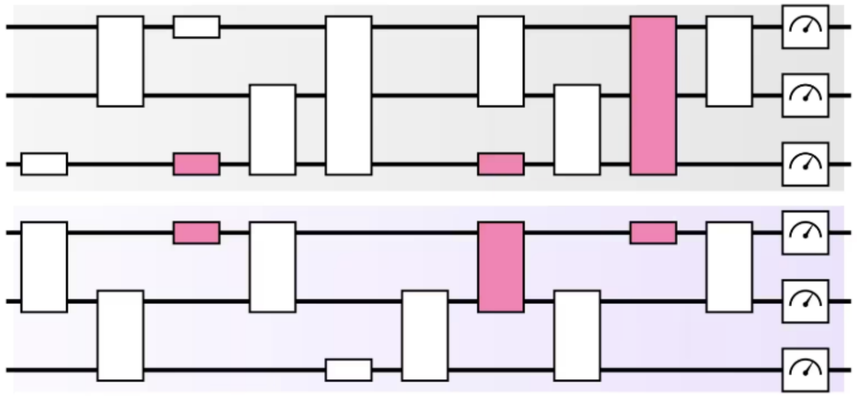
Input:
Circuits, observable

Output:
ISA circuit, observable

---

2. Optimize





A handful of gates, wires, or both are cut, resulting in smaller circuits that are better suited for execution on hardware.

Resulting circuits are executed, and the results are reconstructed through classical post-processing at the cost of increasing the number of shots.

This technique is also used to engineer long-range gates without increasing the 2Q depth.

https://qiskit.github.io/qiskit-addon-cutting/

Running algorithms on quantum hardware

OBP

Circuit cutting

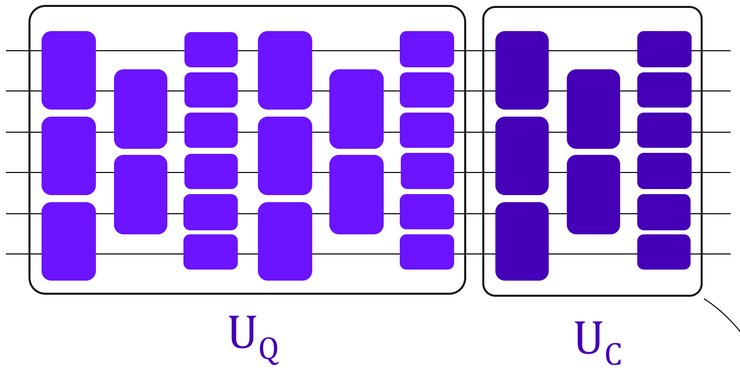
Transpiler

Input:
Circuits, observable

Output:
ISA circuit, observable

---

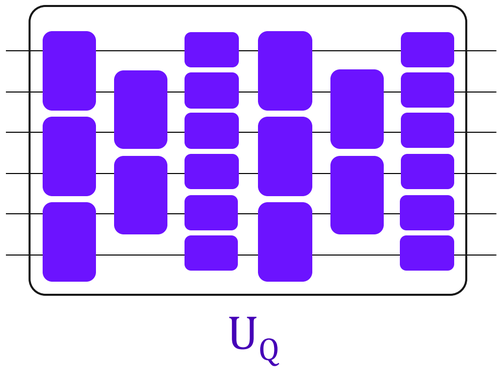
2. Optimize



O

$$\langle\psi| U_Q^\dagger \underbrace{U_C^\dagger O U_C}_{} U_Q |\psi\rangle$$

Classically absorb U_C into O



O'

$$O' = U_c O U_c^\dagger = \sum_P c_P P$$

$$\langle\psi| U_Q^\dagger O' U_Q |\psi\rangle$$

Trading reduced circuit depth for more observables to measure and more classical processing.

https://qiskit.github.io/qiskit-addon-obp/

Running algorithms on quantum hardware

ES and EM

Primitives

Input:
ISA circuit, observable

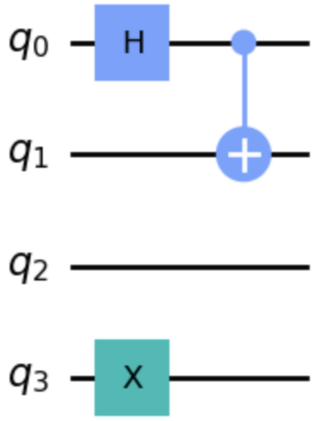
Output:
Expectation value/samples

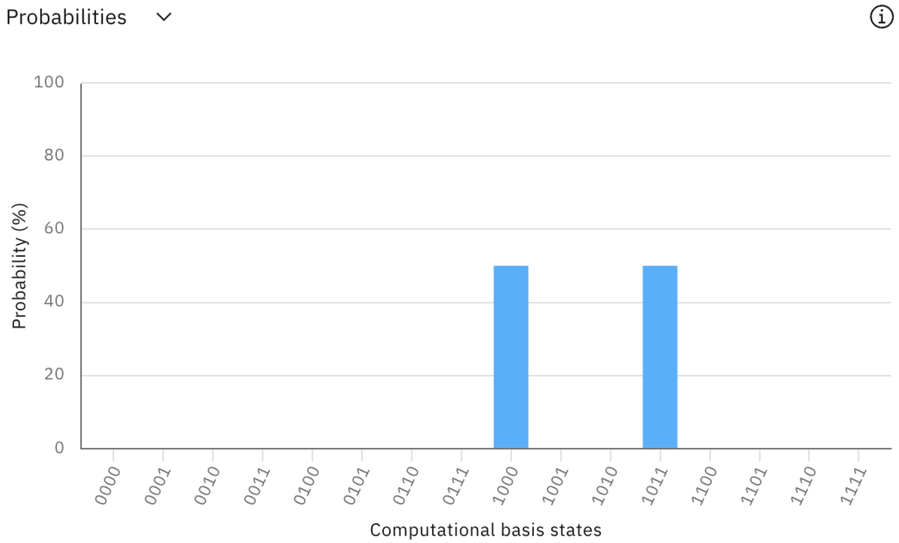
3. Execute

Accessing the output of the quantum circuit

Sampler

Probability of obtaining each basis state by measuring the output of the circuit multiple times.





Running algorithms on quantum hardware

ES and EM

Primitives

Input:
ISA circuit, observable

Output:
Expectation value/samples

3. Execute

Accessing the output of the quantum circuit

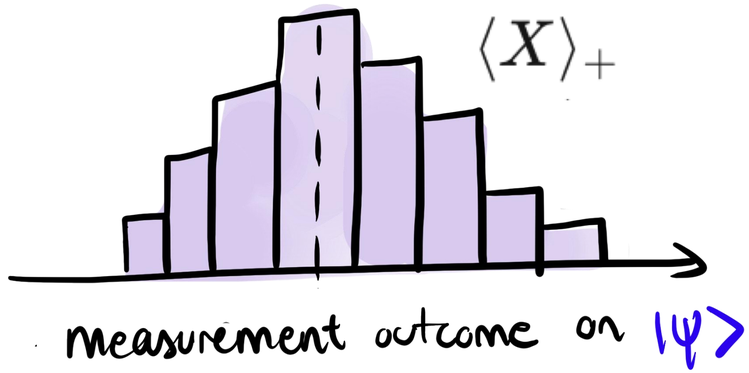
> Estimator
> Expectation value of an observable on a real quantum state by decomposing the observable into operators with known eigenbases.

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

Quantum state

$$\hat{X} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Observable



Running algorithms on quantum hardware

ES and EM

Primitives

Input:
ISA circuit, observable

Output:
Expectation value/samples

3. Execute

Quantum computers are noisy!

We need ways to recover a better signal:

1. Limit the amount of noise
2. Clean the signal by filtering the noise out

This is accomplished by:

- Run modified noisy quantum computations
- Process collected outputs on a classical computer

The primitives include options to implement error suppression and error mitigation.

Running algorithms on quantum hardware

SQD

M3

Quantum info/visualization

Input:
Expectation value/samples

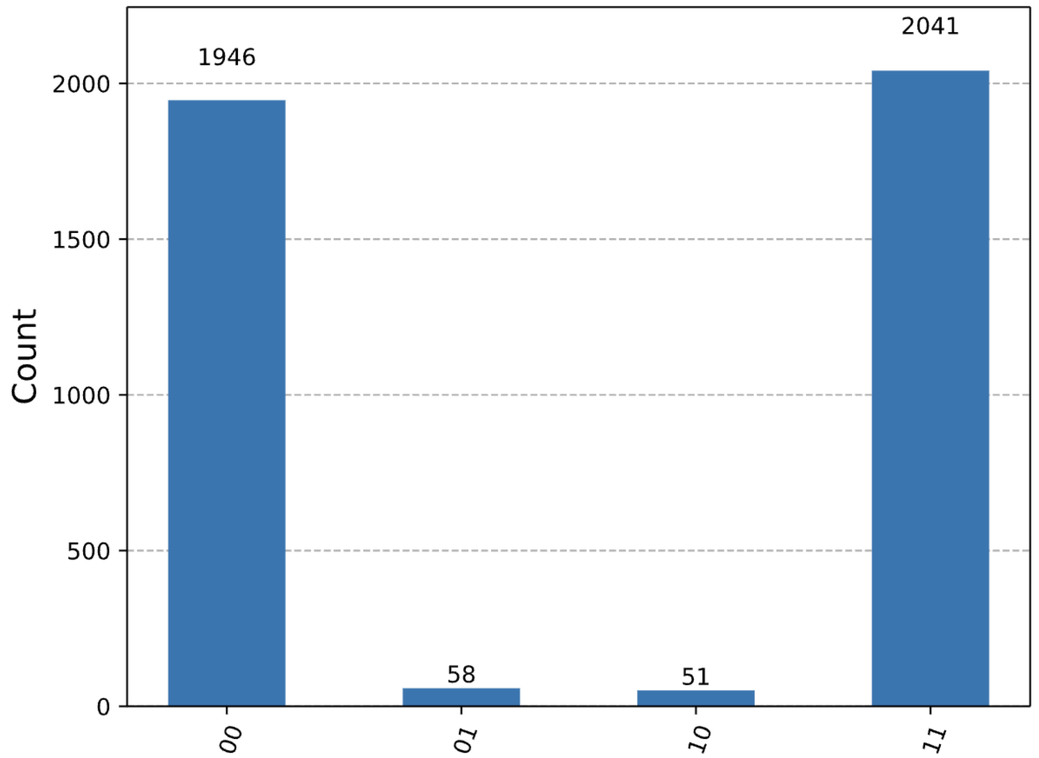
Output:
Data objects/visualizations

---

4. Post-Process

```
plot_histogram(result[0].data.meas.get_counts())
```

Output:



Running algorithms on quantum hardware

SQD

M3

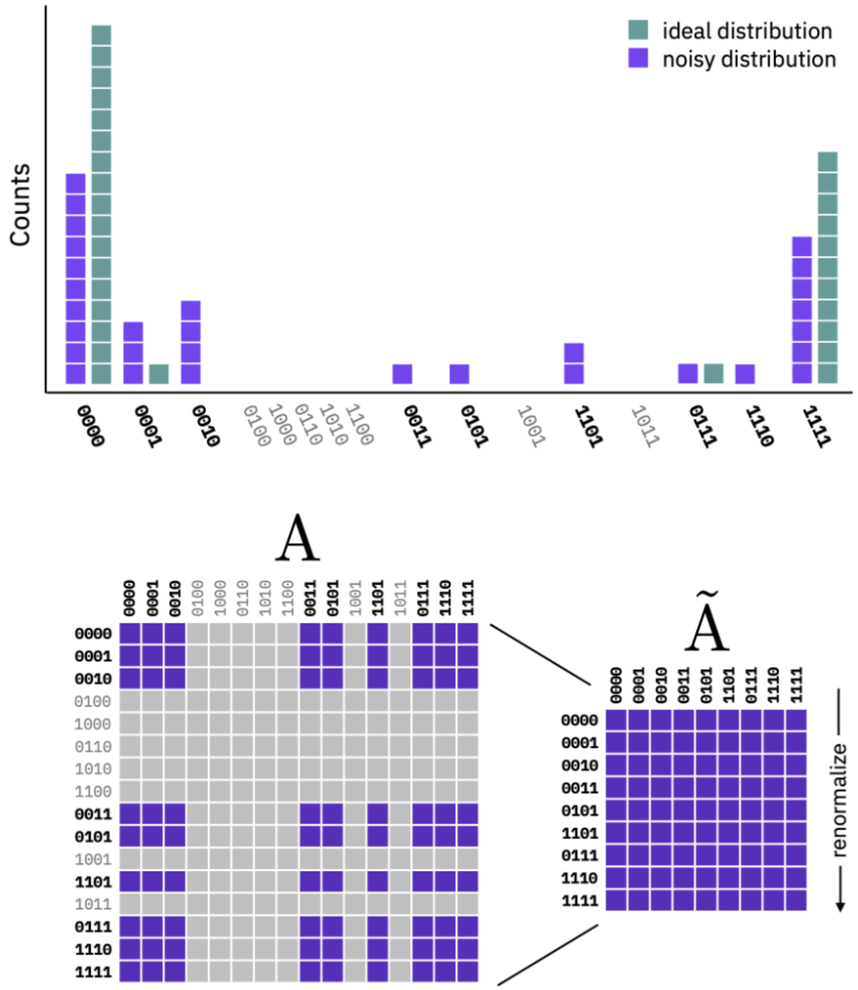
Quantum info/visualization

Input:
Expectation value/samples

Output:
Data objects/visualizations

4. Post-Process

Matrix-free Measurement Mitigation (M3)



M3 works in a reduced subspace defined by the noisy input bitstrings that are to be corrected. Because the number of unique bitstrings can be much smaller than the dimensionality of the full multi-qubit Hilbert space, the resulting linear system of equations is nominally much easier to solve.

https://qiskit.github.io/qiskit-addon-mthree/

Running algorithms on quantum hardware

SQD

M3

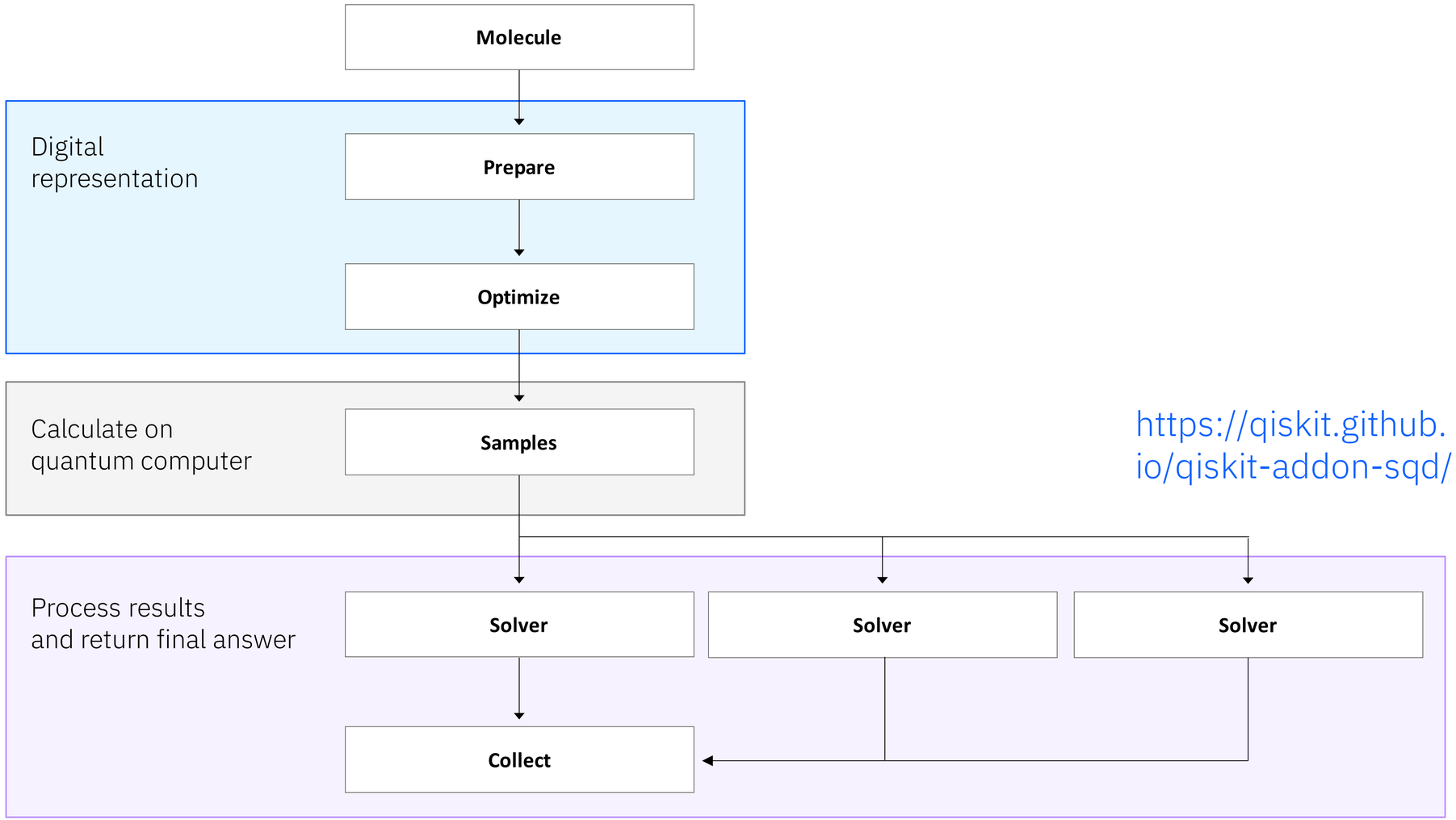
Quantum info/visualization

Input:
Expectation value/samples

Output:
Data objects/visualizations

4. Post-Process

Flow chart of SQD algorithm:

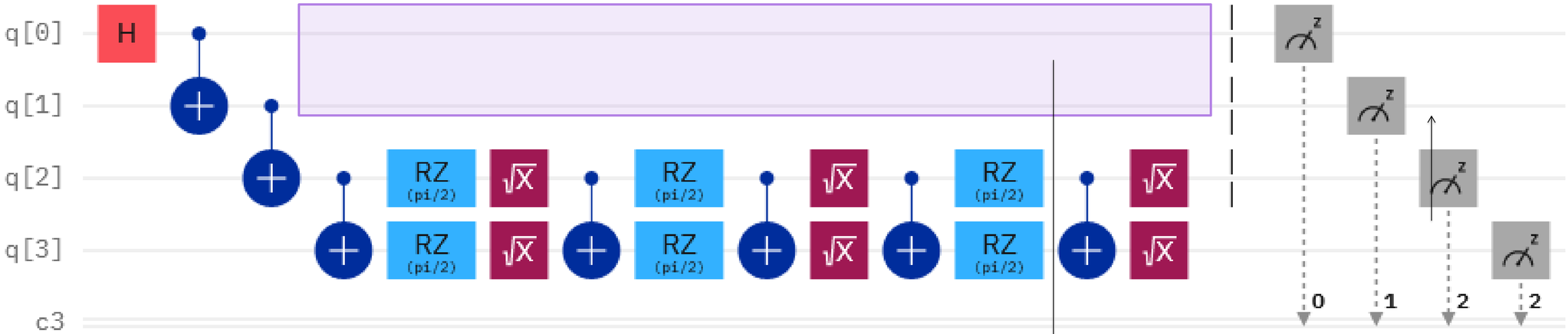


https://qiskit.github.io/qiskit-addon-sqd/

~~Executing on quantum hardware~~

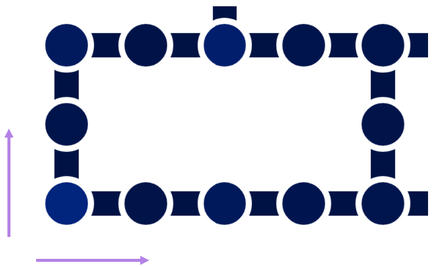
Executing on nosiy quantum hardware

Execution on noisy quantum hardware



Bath/system coupling

Environmental noise: relevant for deep sparse circuits



ibm_torino

OpenQASM 3

Details

133 Qubits	Status:	Online	Median CZ error: 4.384e-3
0.8% EPLG	Total pending jobs:	43 jobs	Median SX error: 3.161e-4
3.8K CLOPS	Processor type ⓘ:	Heron r1	Median readout error: 1.770e-2
	Version:	1.0.16	Median T1: 175.89 us
	Basis gates:	CZ, ID, RZ, SX, X	Median T2: 134.83 us
	Your instance usage:	0 jobs	

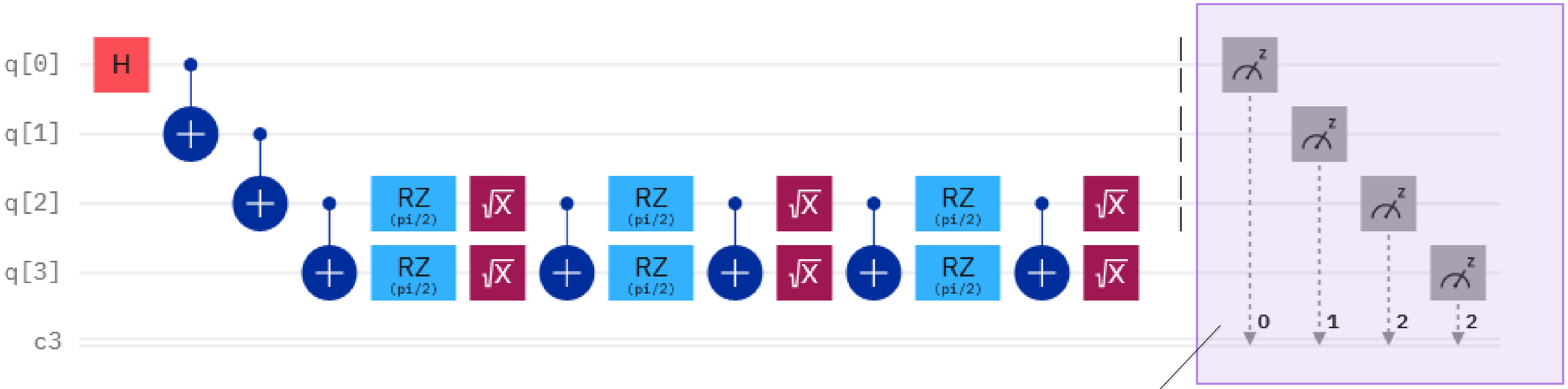
Decoherence

Information loss over time.

Crosstalk

Idle qubits interact with their neighbors. (Difficult to predict but less severe in heron devices!)

Extracting Accurate Results from Noisy Quantum Hardware



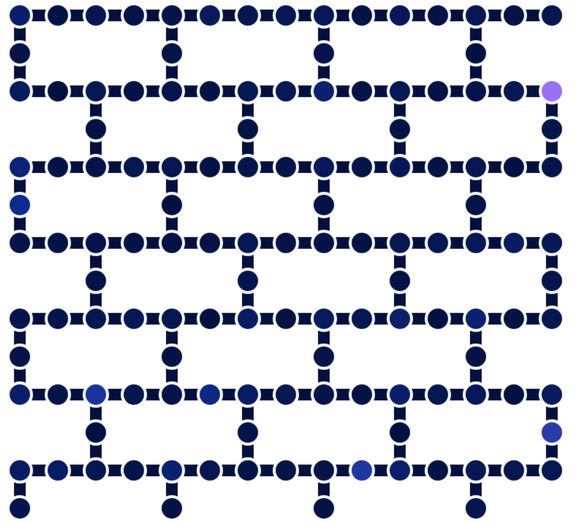
ibm_torino

OpenQASM 3

Details

133 Qubits	Status:	Online	Median CZ error: 4.384e-3
0.8% EPLG	Total pending jobs:	43 jobs	Median SX error: 3.161e-4
3.8K CLOPS	Processor type ⓘ:	Heron r1	Median readout error: 1.770e-2
	Version:	1.0.16	Median T1: 175.89 us
	Basis gates:	CZ, ID, RZ, SX, X	Median T2: 134.83 us
	Your instance usage:	0 jobs	

Readout errors
because of imperfect measurements.
Most important in shallow circuits.



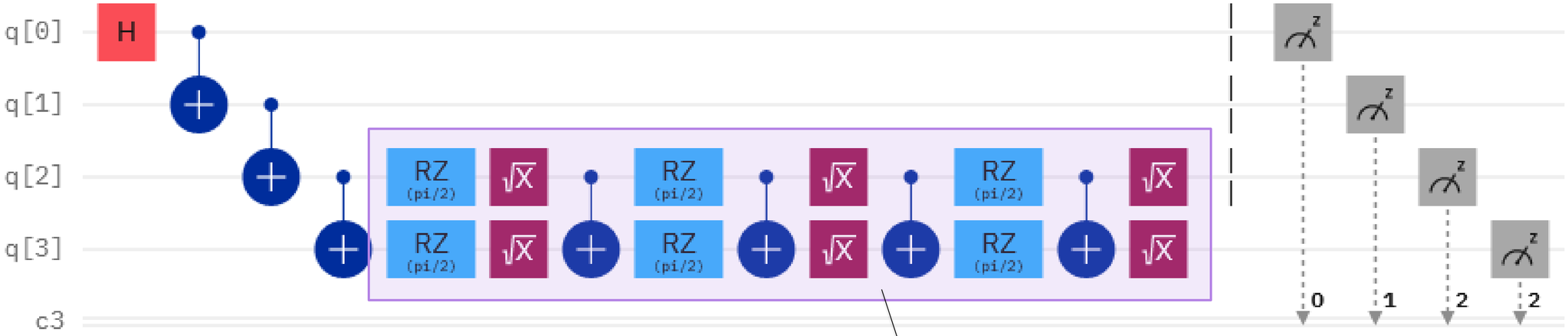
Qubit:

Readout assignment error

Median 1.940e-2

min 3.333e-3 max 4.333e-1

Extracting Accurate Results from Noisy Quantum Hardware



ibm_torino

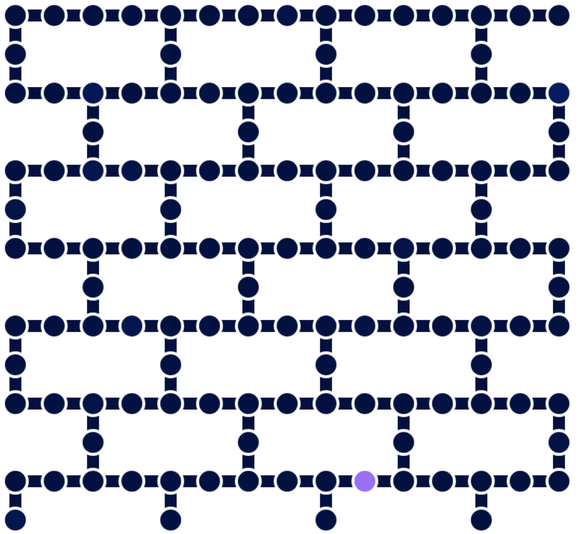
OpenQASM 3

Details

133 Qubits	
0.8% EPLG	
3.8K CLOPS	
Status:	Online
Total pending jobs:	43 jobs
Processor type ⓘ:	Heron r1
Version:	1.0.16
Basis gates:	CZ, ID, RZ, SX, X
Your instance usage:	0 jobs
Median CZ error:	4.384e-3
Median SX error:	3.161e-4
Median readout error:	1.770e-2
Median T1:	175.89 us
Median T2:	134.83 us

Gate errors

Because of imperfect operations on qubits.
(Higher for deep dense circuits with many two-qubit gates).



Qubit: √x (sx) error — Median 3.299e-4 — min 1.335e-4 max 8.469e-2

Connection: CZ error — Median 1.000e+0 — min 1.000e+0 max 1.000e+0

Extracting Accurate Results from Noisy Quantum Hardware

Fault tolerance

Error correction builds up redundancies that allow us to detect and correct errors when they ocurr, leading to essentially error-free logical qubits!

Before fault tolerance…

Error suppression techniques transform a circuit during compilation to minimize noise.

- Dynamical decoupling (DD)
- Pauli Twirling (PT)

Before or during execution (typically)

Additional classical resources dominate

Error mitigation techniques reduce circuit errors by modeling the device noise that was present at the time of execution.

- Twirled Readout Error eXtinction (TREX)
- Zero Noise Extrapolation (ZNE)

After or during execution (typically)

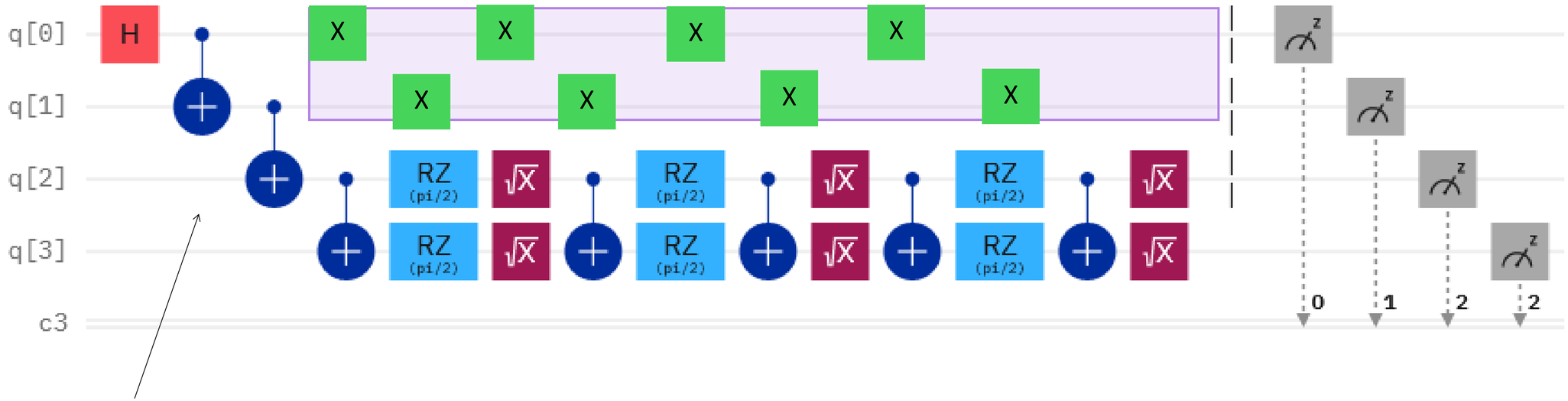
Additional quantum resources dominate

https://www.ibm.com/quantum/blog/quantum-error-suppression-mitigation-correction

Hardware level error suppression

Dynamical decoupling (DD)

~~Cross talk~~



No need to apply gates to qubits which are not initialized.

Hardware level error suppression – how to implement it?

Choose a backend!

```
from qiskit_ibm_runtime import QiskitRuntimeService

service = QiskitRuntimeService()
backend = service.least_busy()
```

✓ 40.8s

Define an object to tweak the options

```
from qiskit_ibm_runtime import SamplerOptions, EstimatorOptions

sampler_options = SamplerOptions(default_shots=1024) #or
estimator_options = EstimatorOptions(default_shots=1024, optimization_level=0, resilience_level=0)
```

✓ 0.0s

We will use Sampler or Estimator V2

```
from qiskit_ibm_runtime import SamplerV2, EstimatorV2

sampler = SamplerV2(backend, options=sampler_options)
estimator = EstimatorV2(backend, options=estimator_options)
```

✓ 0.0s

Sampler options: https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit_ibm_runtime.options.SamplerOptions

Estimator options: https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit_ibm_runtime.options.EstimatorOptions

Hardware level error suppression – Dynamical decoupling

Options	Sub-options	Sub-sub-options	Choices	Default
dynamical_decoupling	enable		`True`/`False`	`False`
	sequence_type		`'XX'`/`'XpXm'`/`'XY4'`	`'XX'`
	extra_slack_distribution		`'middle'`/`'edges'`	`'middle'`
	scheduling_method		`'asap'`/`'alap'`	`'alap'`

```python
from qiskit_ibm_runtime import SamplerOptions, EstimatorOptions

options = SamplerOptions(default_shots=1024)  # or...
options = EstimatorOptions(default_shots=1024, optimization_level=0, resilience_level=0)

## Configure Dynamical Decoupling
options.dynamical_decoupling.enable = True
options.dynamical_decoupling.sequence_type = 'XX'
options.dynamical_decoupling.extra_slack_distribution = 'middle'
options.dynamical_decoupling.scheduling_method = 'alap'
```

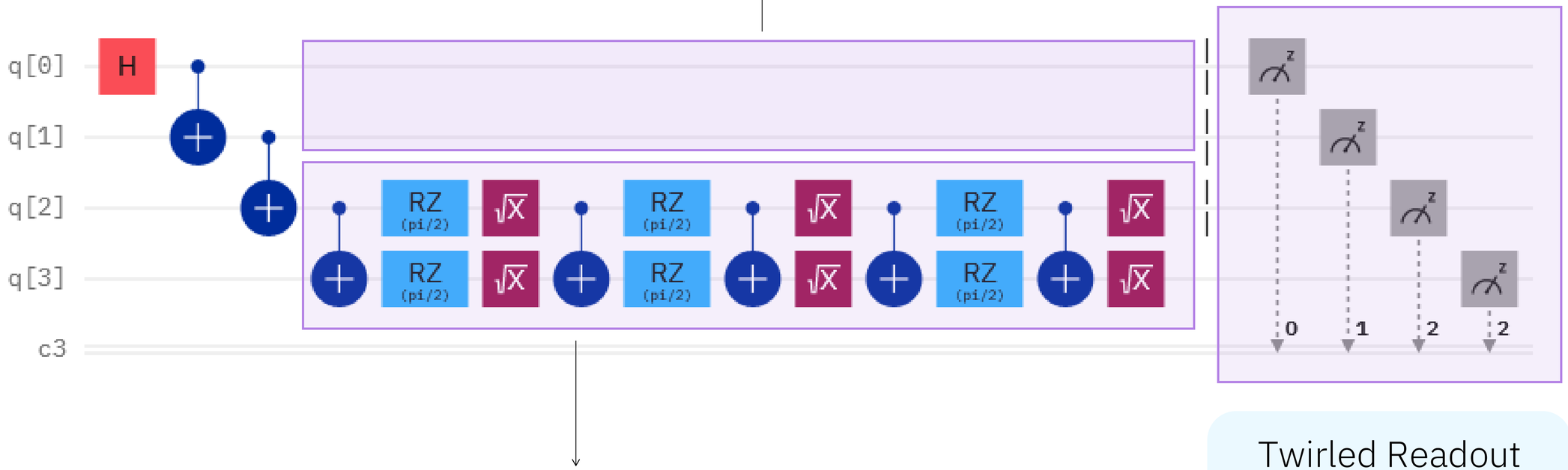
https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit_ibm_runtime.options.DynamicalDecouplingOptions

Extracting Accurate Results from Noisy Quantum Hardware

Dynamical decoupling (DD)

Readout errors

Environmental noise



Gate errors

Twirled Readout Error eXtinction (TREX)

Pauli Twirling (PT)

Zero Noise Extrapolation (ZNE)

~~Executing on quantum hardware~~

Executing on nosiy quantum hardware

Noise manipulation and characterization:

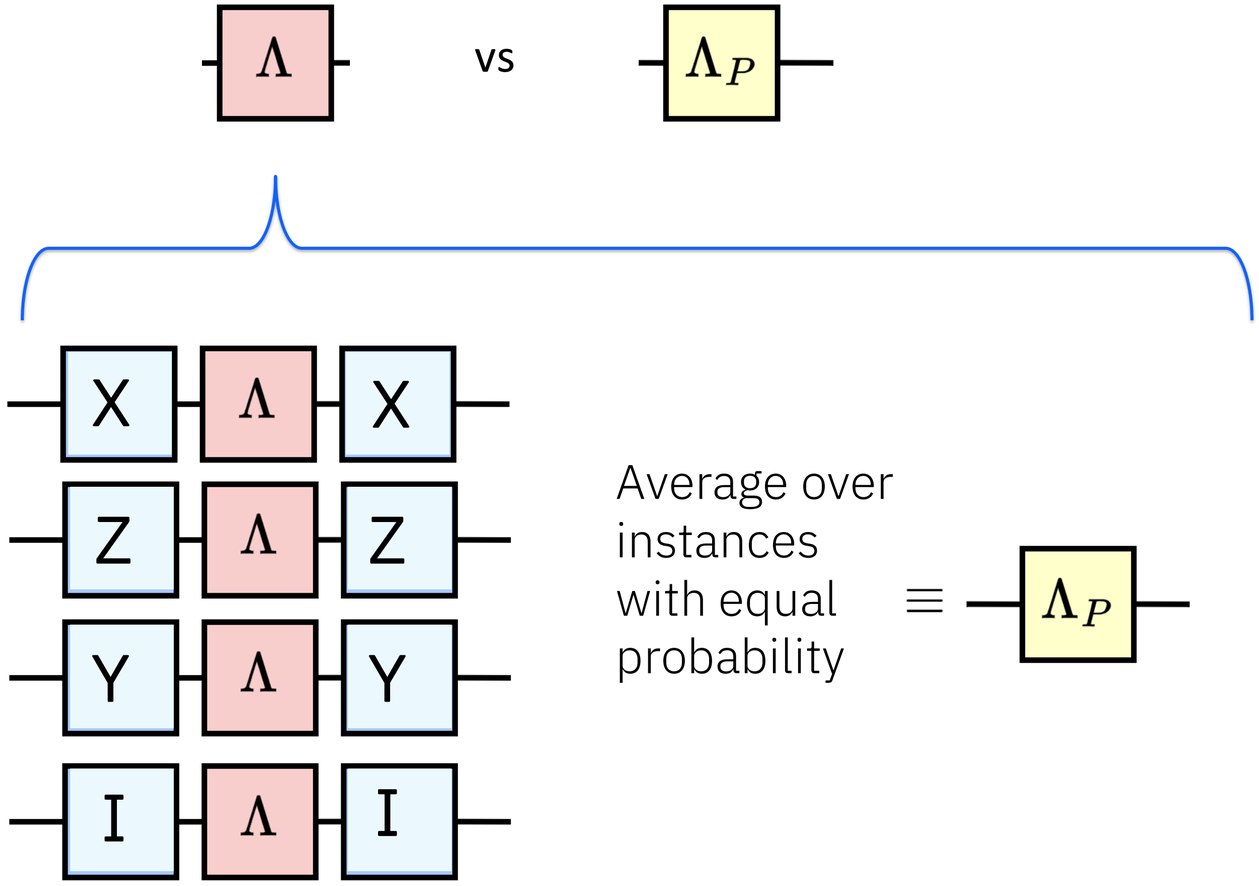
- Pauli Twirling
- Noise learner

+ benchmarking and callibration

Noise characterization and manipulation

Pauli Twirling

Arbitrary noise channels can be converted into Pauli noise.



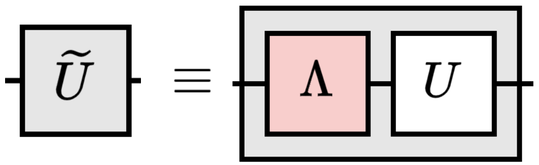
Pauli noise accumulates linearly, in contrast to coherent noise, which accumulates quadratically!

Noise characterization and manipulation

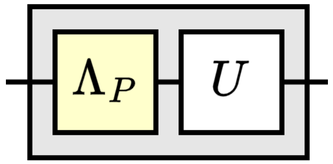
Pauli Twirling

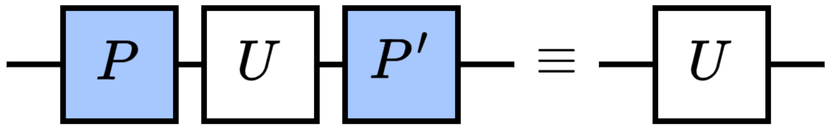
We can consider noisy gates to be associated to a noisy channel Λ.

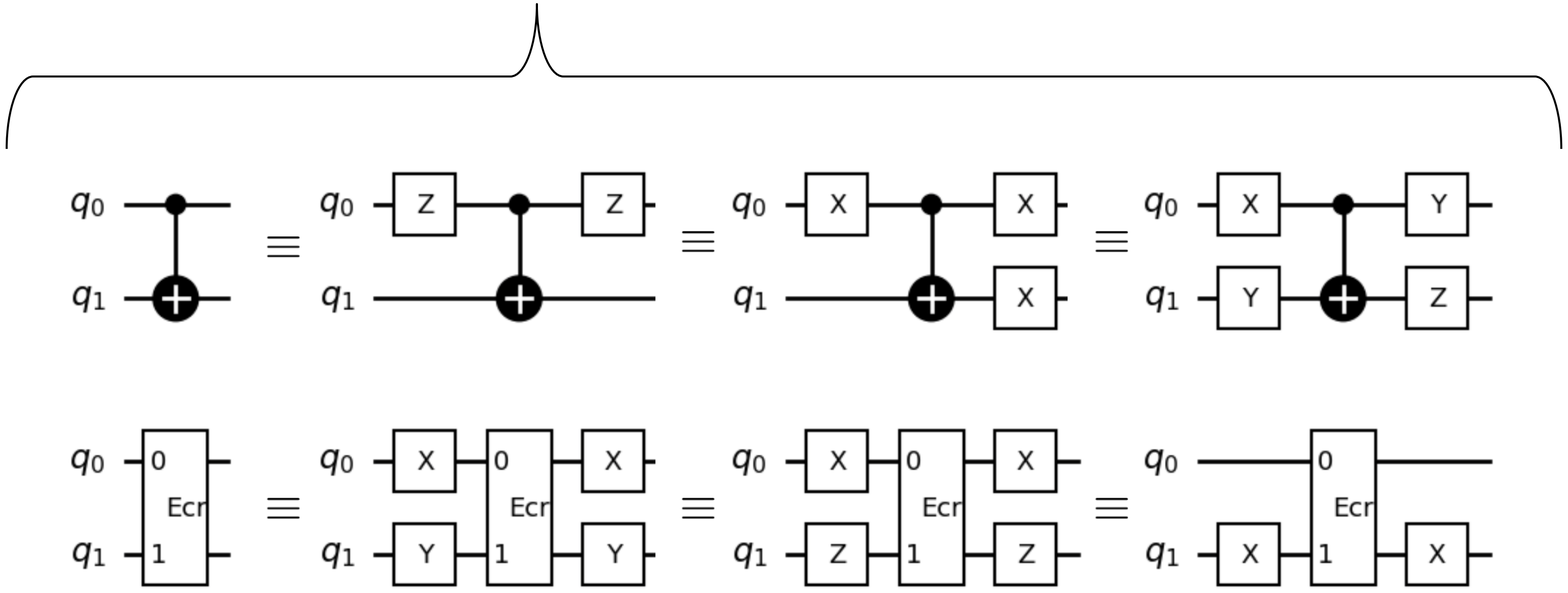
Arbitrary quantum channel



Pauli channel







Set of all possible combinations for a given gate.

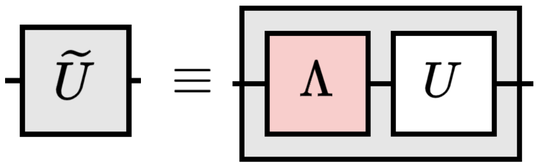
(for Clifford gates 4x4=16 possibilities)

Noise characterization and manipulation

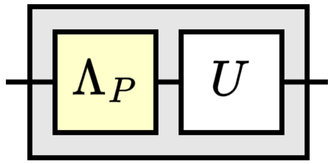
Pauli Twirling

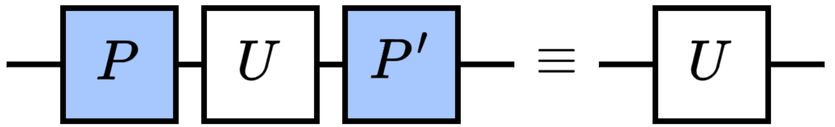
We can consider noisy gates to be associated to a noisy channel $\boldsymbol{\Lambda}$.

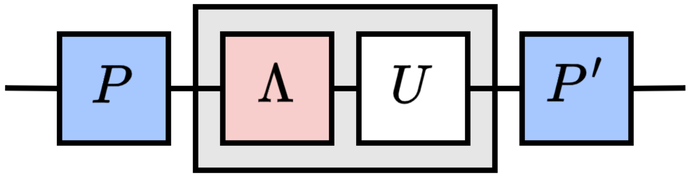
Arbitrary quantum channel



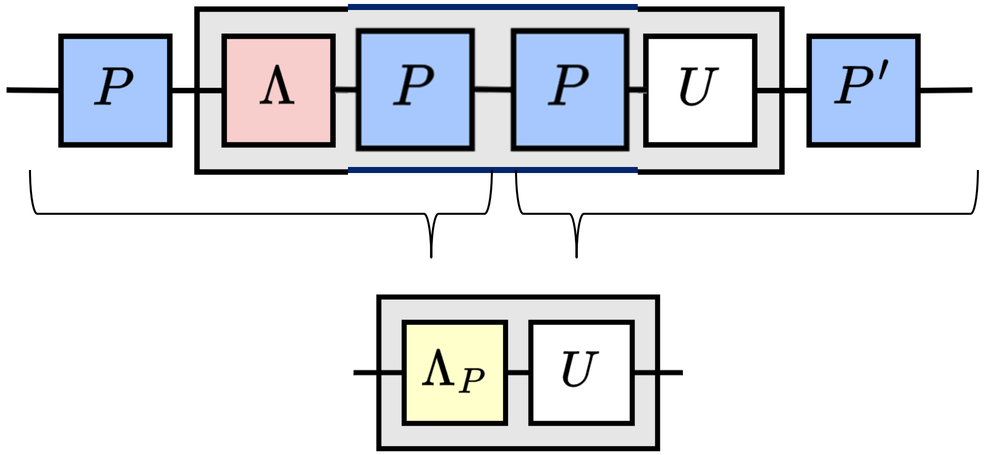
Pauli channel







Average over random instances from the set of possible combinations



Noise characterization and manipulation

Pauli Twirling

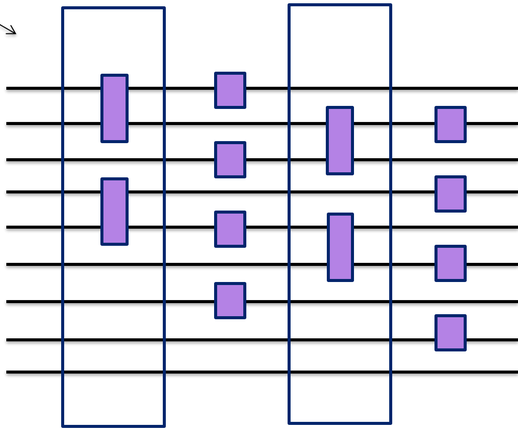
Options	Sub-options	Sub-sub-options	Choices	Default
twirling	enable_gates		`True`/`False`	`False`
	enable_measure		`True`/`False`	`True`
	num_randomizations			`'auto'`
	shots_per_randomization			`'auto'`
	strategy		`'active'`/ `'active-circuit'`/ `'active-accum'`/ `'all'`	`'active-accum'`

(later)

```
from qiskit_ibm_runtime import SamplerOptions, EstimatorOptions

options = SamplerOptions(default_shots=1024)  # or...
options = EstimatorOptions(default_shots=1024, optimization_level=0, resilience_level=0)

## Configure Twirling
options.twirling.enable_gates = True
options.twirling.enable_measure = False
options.twirling.num_randomizations = 'auto'
options.twirling.shots_per_randomization = 'auto'
options.twirling.strategy = 'active-accum'
```



https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit_ibm_runtime.options.TwirlingOptions

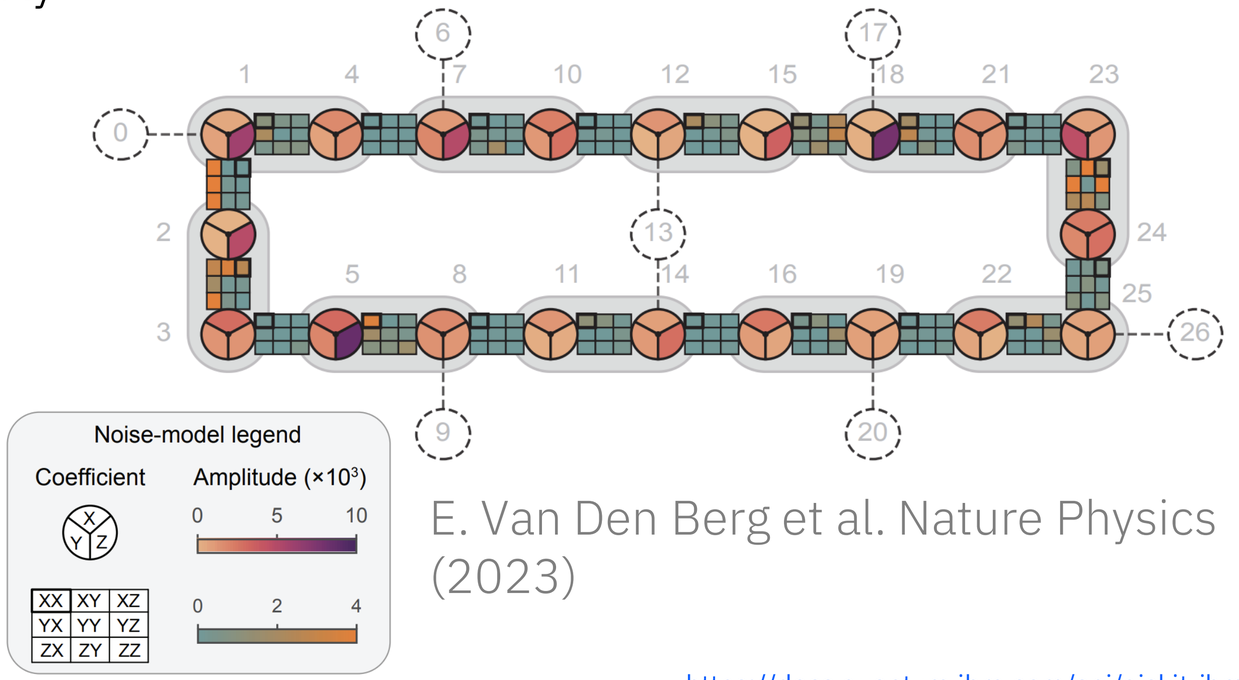
Noise characterization and manipulation

Noise learner

- We can use the noise learner to characterize the noise affecting the 2Q gate layers in a given circuit.
- It learns the noise assuming a sparse Pauli-Lindblad noise model.

The parameters of the resulting model scale linearly with the number of qubits, therefore the model is efficiently represented and easy to learn.

Caveat: The learning requires more circuit executions and therefore adds overhead!



E. Van Den Berg et al. Nature Physics (2023)

https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/0.26/qiskit_ibm_runtime.noise_learner.NoiseLearner

Noise characterization and manipulation

Noise learner

```python
from qiskit_ibm_runtime.noise_learner import NoiseLearner
from qiskit_ibm_runtime.options import NoiseLearnerOptions

# set the options
options = NoiseLearnerOptions()
options.layer_pair_depths = [0, 1, 2, 4, 16, 32]
options.max_layers_to_learn = 4
options.num_randomizations = 32
options.shots_per_randomization = 128
options.twirling_strategy = "active-accum"


# run the noise learner job
learner = NoiseLearner(backend, options)
job = learner.run(circuits)
```


http://docs.quantum.ibm.com/api/qiskit-ibm-runtime/options-noise-learner-options

~~Executing on quantum hardware~~

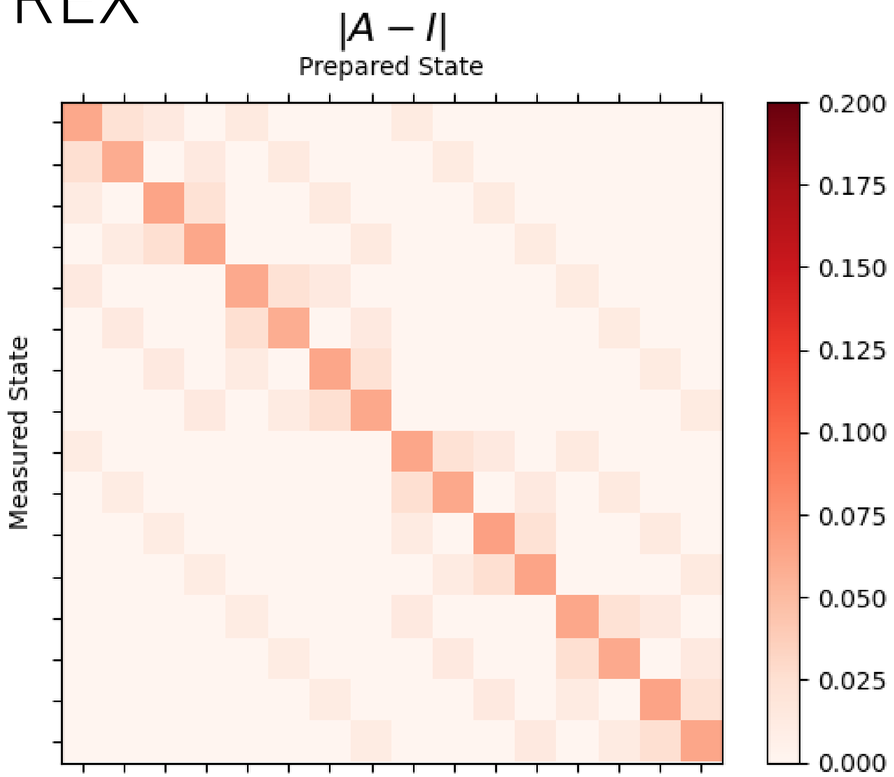
Executing on nosiy quantum hardware

Undoing the effect of noise:

- TREX
- Zero Noise Extrapolation (ZNE/PEA)
- Probabilistic Error Cancellation (PEC)

Undoing the effect of noise

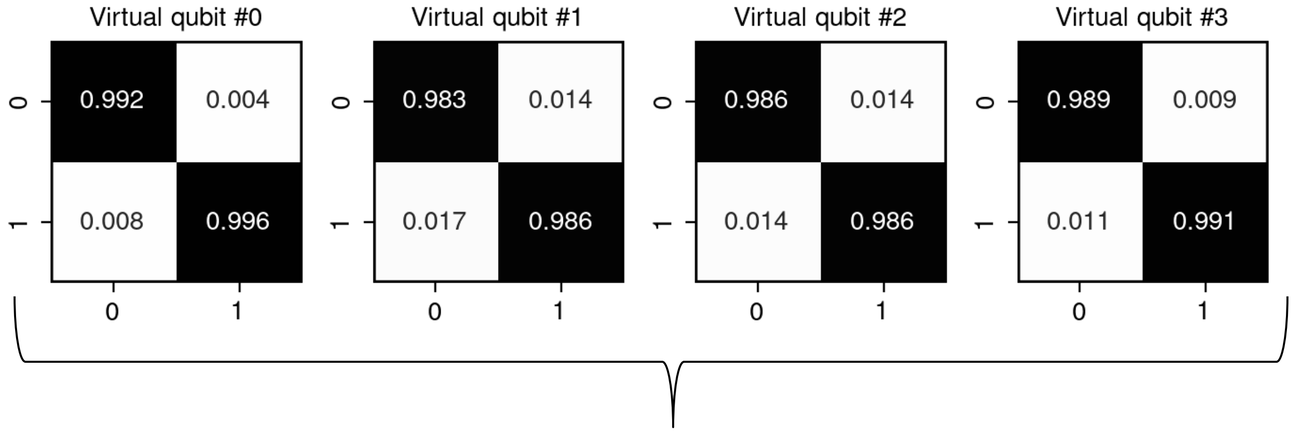
TREX



Readouts errors cause the wrong states to be measured.

This is depicted in the readout-error transfer matrix.

In scenarios when the noise is not correlated between qubits, measurement errors can be measured per qubit and the full transfer matrix is then reconstructed as a tensor product.



$$2^N \times 2^N$$

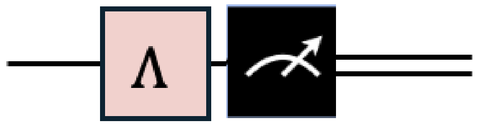
The inverse of the transfer matrix can be used for error mitigation, but it cannot be obtained efficiently in general!

Undoing the effect of noise

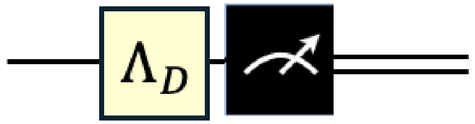
TREX

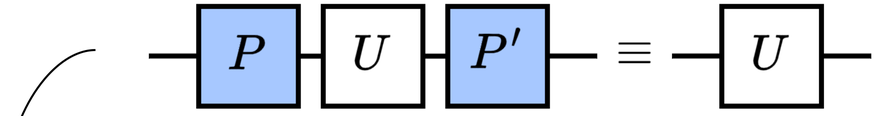
Via **measurement twirling we can diagonalize the** readout-error transfer matrix.

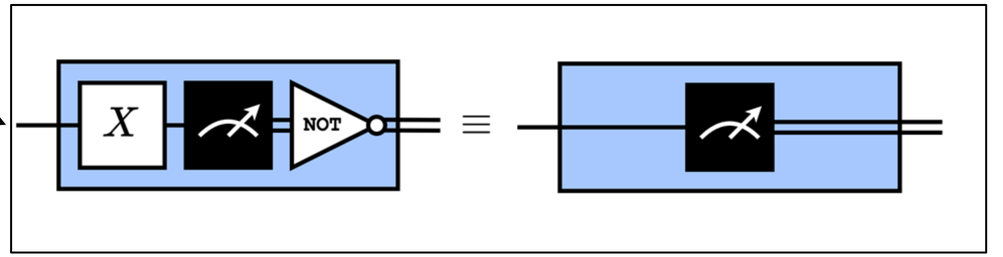
Arbitrary transfer matrix



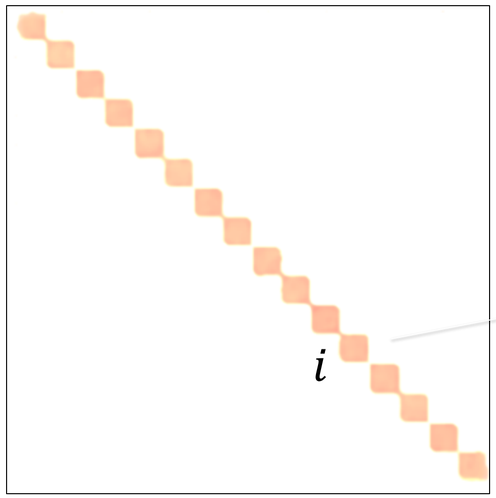
Diagonal transfer matrix







Average over random bitflips that are undone in classical postprocessing.



Then, we need to run calibration circuits to get the inverse of the (diagonal) readout error transfer matrix.

$$< O_i > = \frac{< \widetilde{O_i} >}{E_i}$$

Only valid for expectation values!

Undoing the effect of noise

TREX

Options	Sub-options	Sub-sub-options	Choices	Default
resilience	measure_mitigation		`True`/`False`	`True`
	measure_noise_learning	num_randomizations		`32`
		shots_per_randomization		`'auto'`

```
from qiskit_ibm_runtime import EstimatorOptions

options = EstimatorOptions(default_shots=1024, optimization_level=0, resilience_level=0)

## Configure TREX
options.resilience.measure_mitigation = True
options.resilience.measure_noise_learning.num_randomizations = 32
options.resilience.measure_noise_learning.shots_per_randomization = 'auto'

options.twirling.enable_measure = True  # Automatically set by TREX
```

https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit_ibm_runtime.options.ResilienceOptionsV2

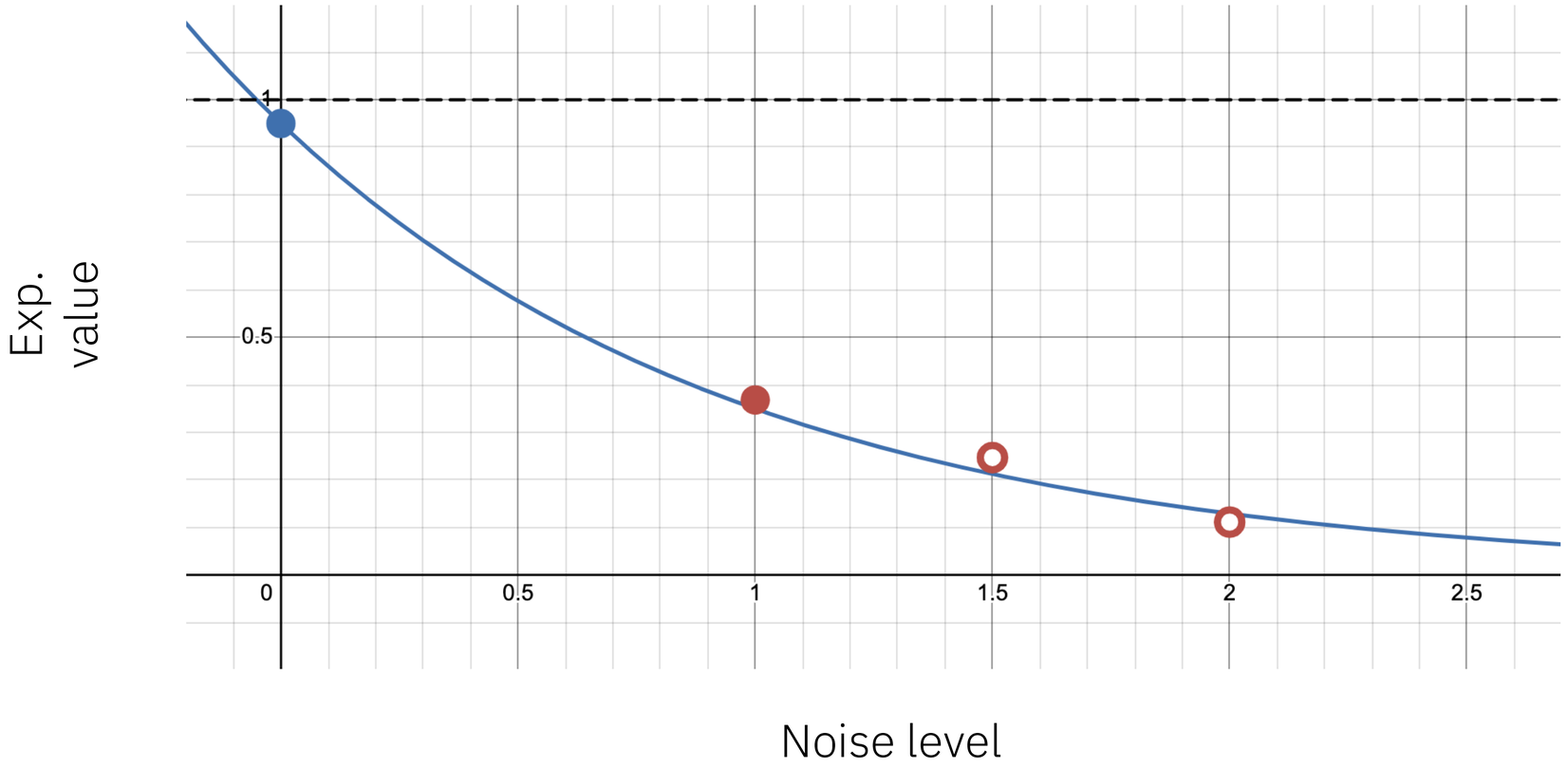
Undoing the effect of noise

ZNE

Measures the effects of amplified noise to infer what the results would look like in the absence of noise.

1. Noise amplification:
   the original circuit unitary is executed at different levels of noise.

2. Extrapolation:
   the zero-noise limit is inferred from the noisy expectation-value results.

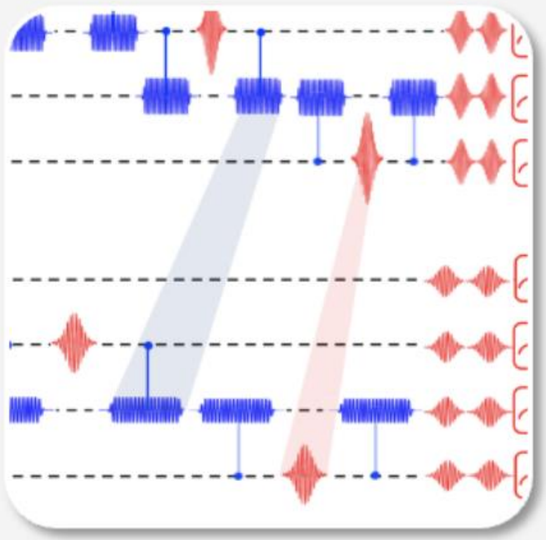


Undoing the effect of noise

ZNE

Step 1: Noise amplification

Pulse stretching	Gate folding	Probabilistic error amplification
Scale pulse duration via calibration	Repeat gates in identity cycles $U \mapsto U(U^{-1}U)^{\lambda-1}/2$	Add noise via sampling Pauli channels



Kandala et al. Nature (2019)

It’s an analogue technique.

It assumes noise is proportional to pulse duration.

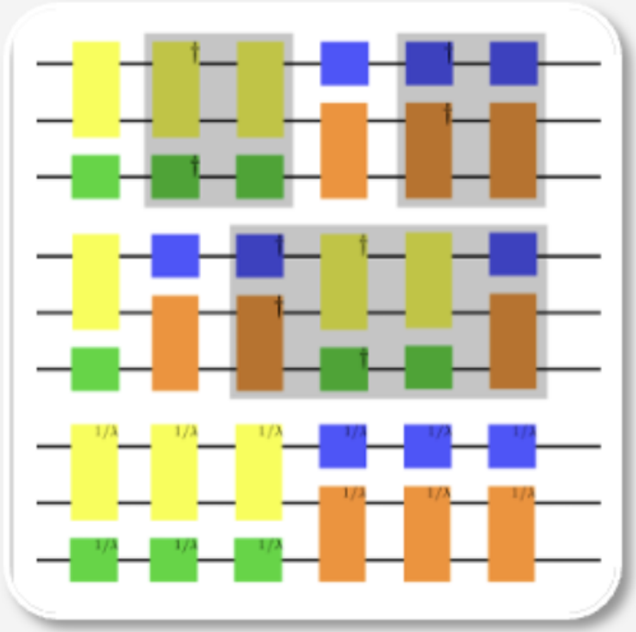
It requires costly pulse level calibration of the hardware.

Undoing the effect of noise

ZNE

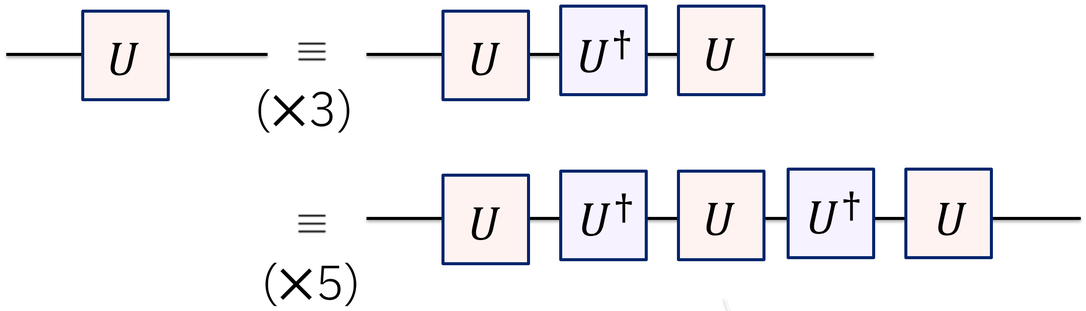
Step 1: Noise amplification

Pulse stretching	Gate folding	Probabilistic error amplification
Scale pulse duration via calibration	Repeat gates in identity cycles $U \mapsto U(U^{-1}U)^{\lambda-1}/2$	Add noise via sampling Pauli channels



Shultz et al. PRA (2022)

It is a heuristic approach but offers a good trade-off between result quality and resource requirements.



But it is limited by circuit depth!

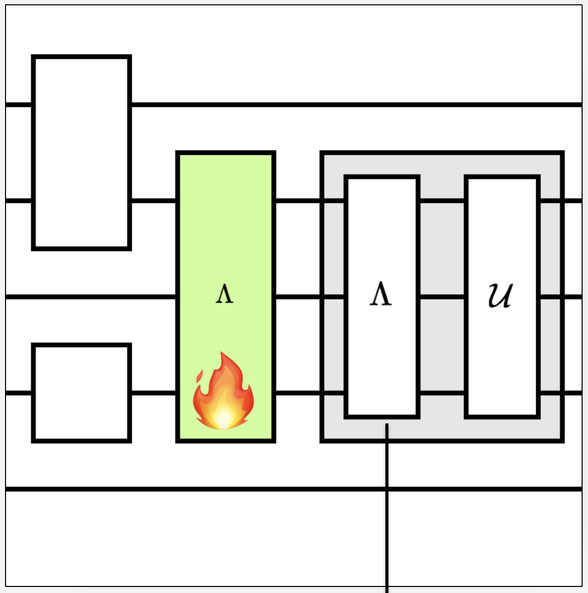
especially powerful when $U \equiv U^\dagger$

Undoing the effect of noise

ZNE

Step 1: Noise amplification

Pulse stretching	Gate folding	Probabilistic error amplification
Scale pulse duration via calibration	Repeat gates in identity cycles $U \mapsto U(U^{-1}U)^{\lambda-1}/2$	Add noise via sampling Pauli channels



Li & Benjamin PRX (2017)

It has general applicability and strong theoretical backing.

Used in the utility paper
Y. Kim et al. Nature (2023).

But it requires learning circuit-specific noise.

Idea: we want to learn the noise channel of certain layers of gates and reproduce it.

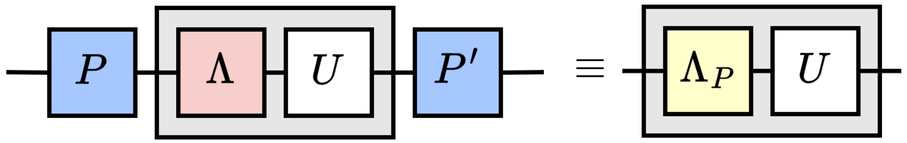
Undoing the effect of noise

ZNE

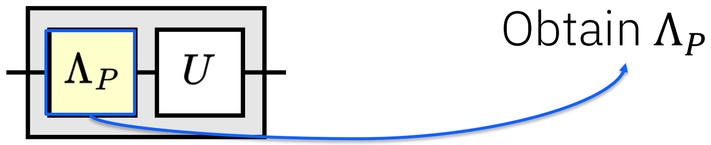
Step 1: Noise amplification

Pulse stretching	Gate folding	Probabilistic error amplification
Scale pulse duration via calibration	Repeat gates in identity cycles $U \mapsto U(U^{-1}U)^{\lambda-1}/2$	Add noise via sampling Pauli channels

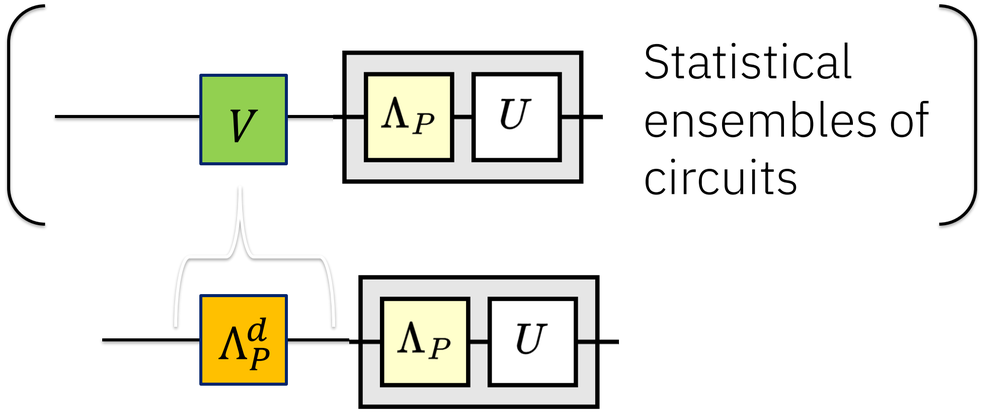
Step 1. Use Pauli Twirling to transform the noise channels to Pauli noise, which is easier to learn.



Step 2. Noise learning.



Step 3. Noise injection using random unitaries.



Undoing the effect of noise

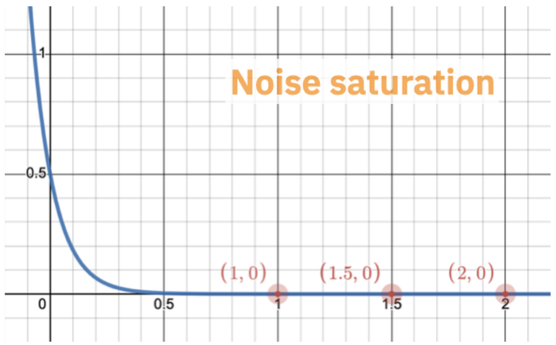
ZNE

Step 2. Extrapolation

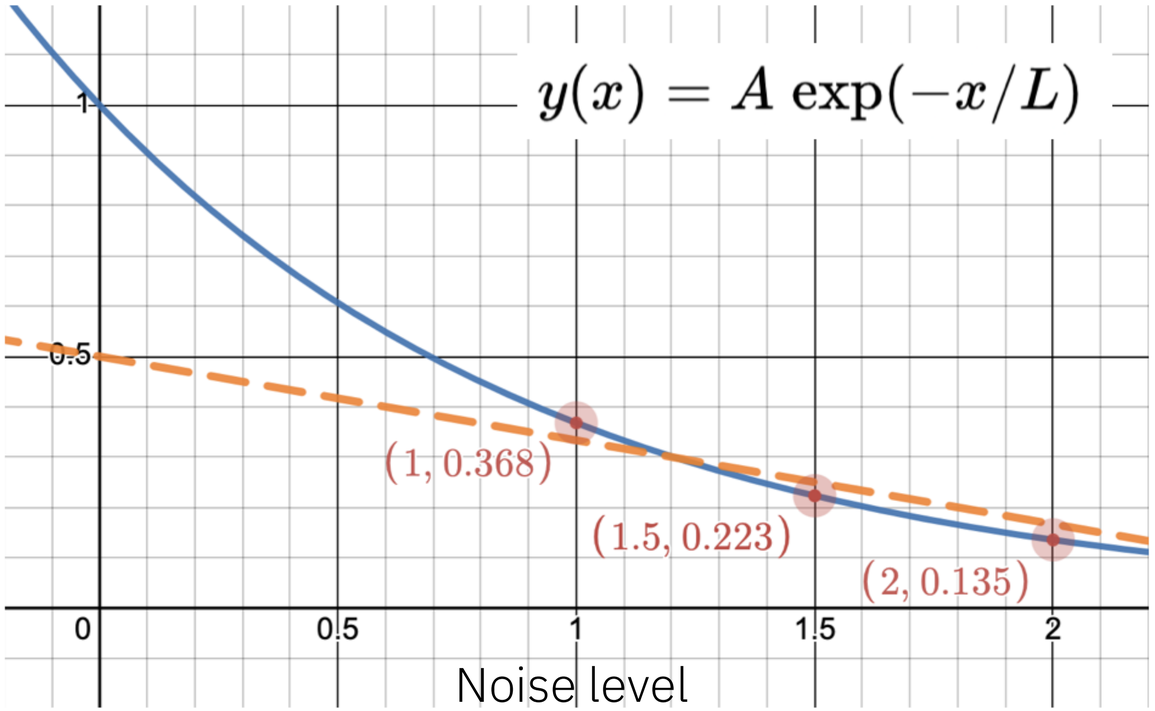
Theoretical/experimental results predict exponential decay in observed expectation values

Exponential extrapolation mitigates aggressively but is unstable because we don't know the scale.

$A = \pm 0.5 \; ; \; L = 0.1$



$A = 1.0 \; ; \; L = 1.0$



Polynomial extrapolation is stable but mitigates worse, since it retains the scale of the noisy data.

Undoing the effect of noise - ZNE

Options	Sub-options	Sub-sub-options	Choices	Default
resilience	zne_mitigation		`True`/`False`	`False`
	zne	noise_factors		`(1, 1.5, 2) for PEA, and (1, 3, 5) otherwise`
		amplifier	`'gate_folding'`/ `'gate_folding_front'`/ `'gate_folding_back'`/ `'pea'`	`gate_folding`
		extrapolator	`'exponential'`/ `'linear'`/ `'double_exponential'`/ `'polynomial_degree_(1 <= k <= 7)'`	(`'exponential'`, `'linear'`)

```python
from qiskit_ibm_runtime import EstimatorOptions

options = EstimatorOptions(default_shots=1024, optimization_level=0, resilience_level=0)

## Configure ZNE
options.resilience.zne_mitigation = True
options.resilience.zne.noise_factors = (1, 3, 5)  [Can have fractional noise factors, e.g. (1, 1.5, 2)]
options.resilience.zne.amplifier = 'gate_folding'
options.resilience.zne.extrapolator = ('exponential', 'linear')
```

Choosing the right noise factors and extrapolator is tricky!

https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit_ibm_runtime.options.ResilienceOptionsV2

Undoing the effect of noise

PEC

The effect of an ideal target circuit can be expressed as a linear combination of noisy circuits that are implementable in practice:

$$\mathcal{O}_{\text{ideal}} = \sum_i \eta_i \mathcal{O}_{noisy,i}$$

Unlike ZNE, it produces bias free expectation values at the cost of increased variance. To account for this, we must take γ^2 times more shots, where:

$$\gamma = \sum_i |\eta_i| \geq 1.$$

Caveat: γ scales exponentially with the depth of the circuit

Main error Mitigation techniques - PEC

Options	Sub-options	Sub-sub-options	Choices	Default
	pec_mitigation		`True`/`False`	`False`
	pec	max_overhead		`None/ integer >1`
		noise_gain		`auto/ float in the range [0, 1] auto`

```python
from qiskit_ibm_runtime import EstimatorOptions

options = EstimatorOptions(default_shots=1024, optimization_level=0, resilience_level=0)

## Configure PEC
options.resilience.pec_mitigation = True
options.resilience.pec.max_overhead = 100
```

https://docs.quantum.ibm.com/api/qiskit-ibm-runtime/qiskit_ibm_runtime.options.PecOptions

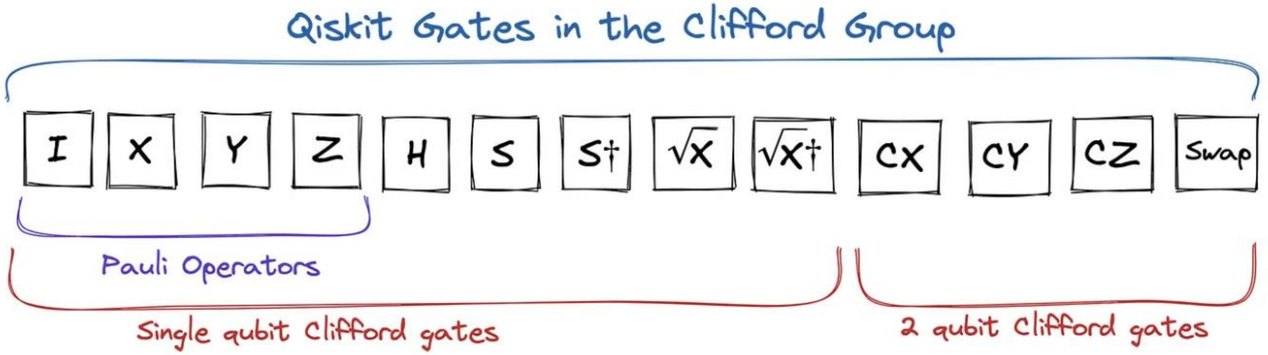
~~Executing on quantum hardware~~

Executing on nosiy quantum hardware

Other strategies to address noise:

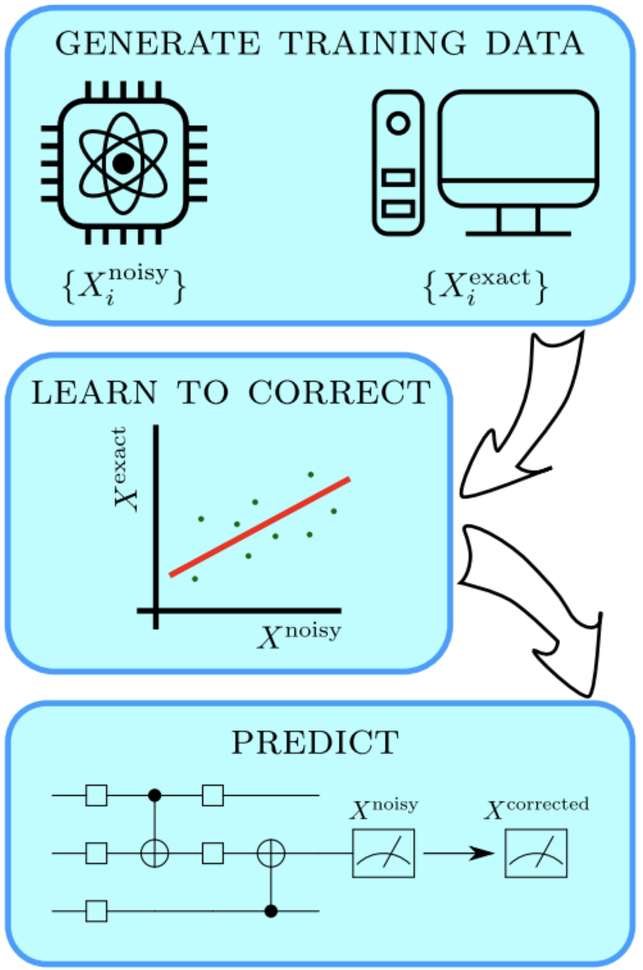
- Validation strategies
- Post-selection and configuration recovery: stepping stones toward error correction

Other strategies to address noise



Clifford circuits are composed of Clifford gates. They can be efficiently simulated on a classical computer but they are not universal.

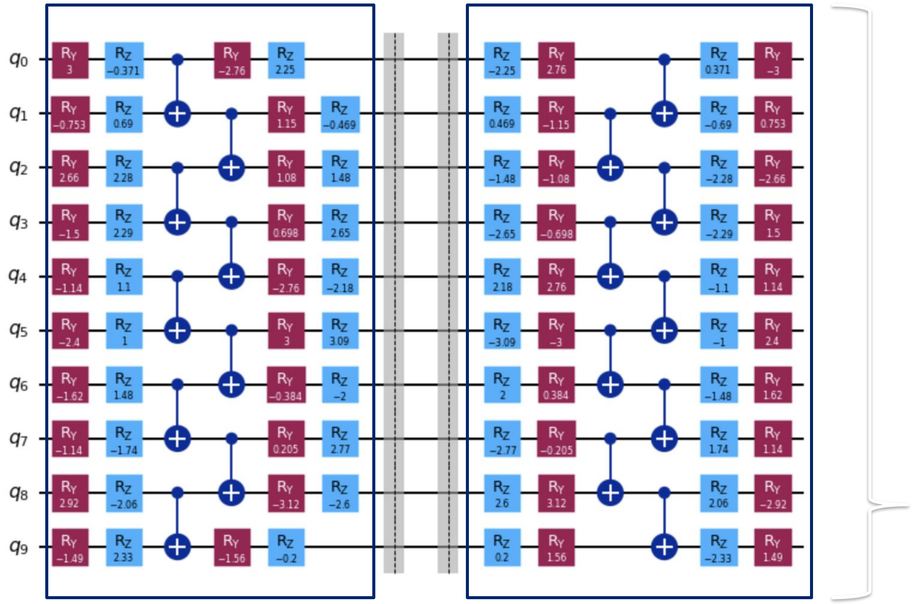
Clifford Data Regression uses cliffordized circuits to generate training data for an error mitigation model.



Quantum 5, 592 (2021)

Other strategies to address noise

Mirror circuits are offer a way to replace a circuits with unknown outcome with another circuit which preserves the overall internal structure and richness, but whose final answer is trivially known.



We expect the zero state

$$\langle \hat{O} \rangle_{\text{meas}} = (1 - \eta_O) \langle \hat{O} \rangle_{\text{pred}}.$$

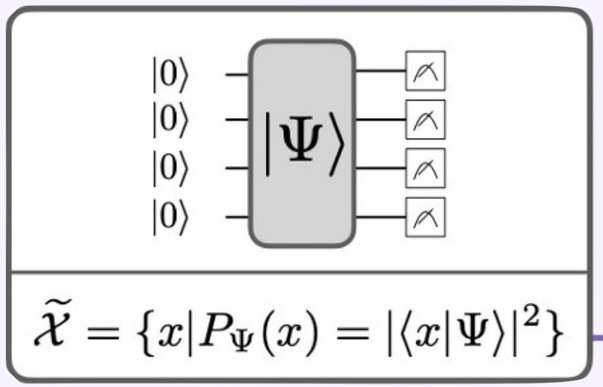
Operator Decoherence Renormalization

Uses a callibration mirror circuit to mitigate the noise in the real circuit.

PRX Quantum 5, 020315 (2024)

Post-selection and configuration recovery

Sampler()



We sample from a circuit in the computational basis.

$$\tilde{\mathcal{X}} = \left\{ \mathbf{x} \mid \mathbf{x} \sim \tilde{P}_\Psi(\mathbf{x}) \right\}$$

1001, 1111
1000, 0111

The bitstrings $\mathbf{x} \in \{0, 1\}^M$ represent solutions to our problem.

In some cases, they must fulfill certain symmetries imposed by the problem we want to solve. For example:

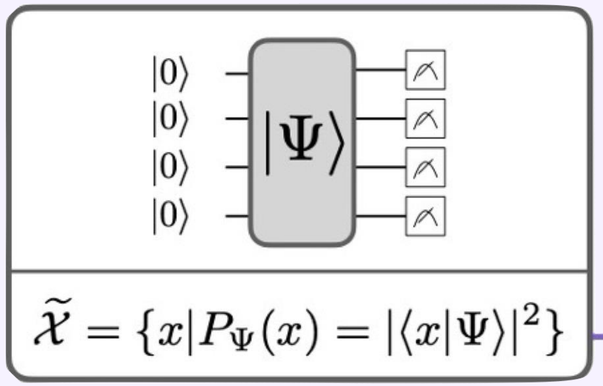
- electronic configurations with fixed number of particles

Noise alters the distribution from its ideal form and generates the noisy set of configurations.

Configuration Recovery

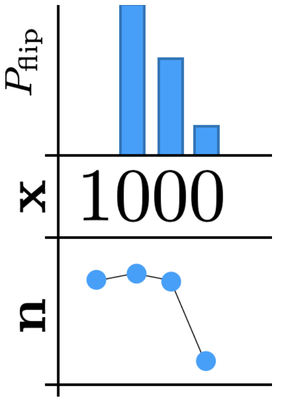
Post-selection and configuration recovery

Sampler()



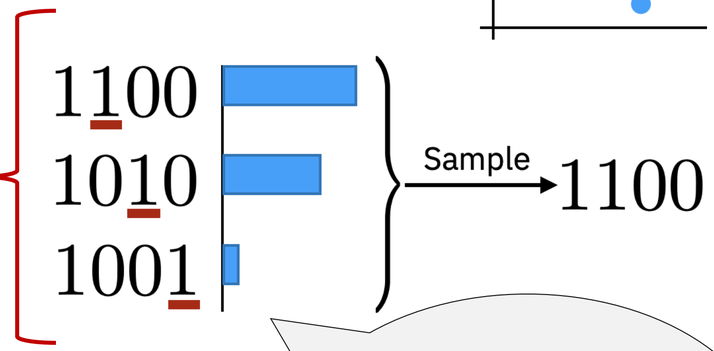
Configuration recovery using particle number symmetry:

For N=2, 1000 is not in the right sector.



Flip bits according to given probability distribution

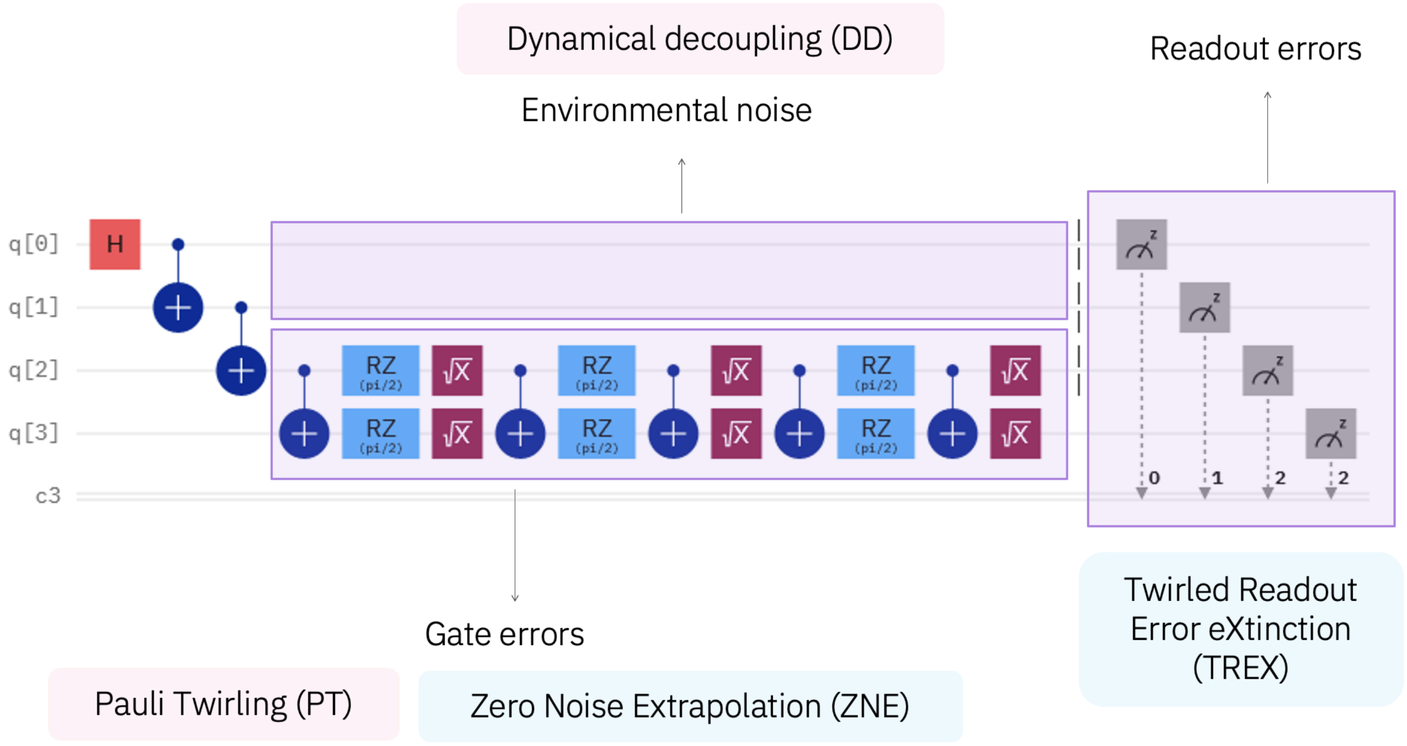
1100
1000
0110



|Nx – N | bits are sampled to be flipped

Post-selection can be implemented in many different ways depending on the symmetries of the problem at hand.

Summary & conclusions



- Learning the noise of the device
- Probabilistic Error Cancellation
- Alternative validation strategies
  - Mirror circuits
  - Clifford circuits
- Post-selection and configuration recovery: stepping stones toward error correction

+ benchmarking and callibration

QGSS, 2025

Practical quantum techniques

Joana Fraxanet
Quantum Algorithm
Engineering team

IBM **Quantum**