# State Tomography Using Classical Shadows

Julia Guignon
January 16, 2025 — Ecole Polytechnique Fédérale de Lausanne

# Challenge: how to know a quantum state

- number of parameters that characterize a quantum state grows exponentially with the dimension of the system
- a measurement is destructive and gives only probabilistic outcomes
- measuring a single parameter requires repeated measurements of many identically prepared samples

$\Longrightarrow$ requires an exponential number of measurements

# Quantum state tomography

# Main idea

- any $n$-qubit state can be expressed as a linear combination of Pauli strings
  $\longrightarrow$ measure the expectation values for each possible string and reconstruct the state
- requires $3^n$ measurements to reconstruct the state fully
- different possible approaches to reconstruct the state: linear, Maximum Likelihood, Bayesian inference

# Implementation

1. Get all the possible Pauli strings of length $n$.
2. For each possibility, measure and report the expectation value.
3. Reconstruct the state: linear reconstruction.
4. ~~Hope for the best~~

```
def contruct_rho_lin(n, expect_val, measurements, valid):
    rhotilde = np.sum([scalar*mat for scalar, mat in
     ↪  zip(list(expect_val.values()),    list(map(get_pauli_str_mat,
     ↪  measurements.keys()))))], axis=0)/2**n # /2**n for the normalization of
     ↪  states
    return rhotilde.todense()
```

# Results?


A new foe has appeared!
CHALLENGER APPROACHING
QiskitError: 'Input
quantum state
is not a valid'

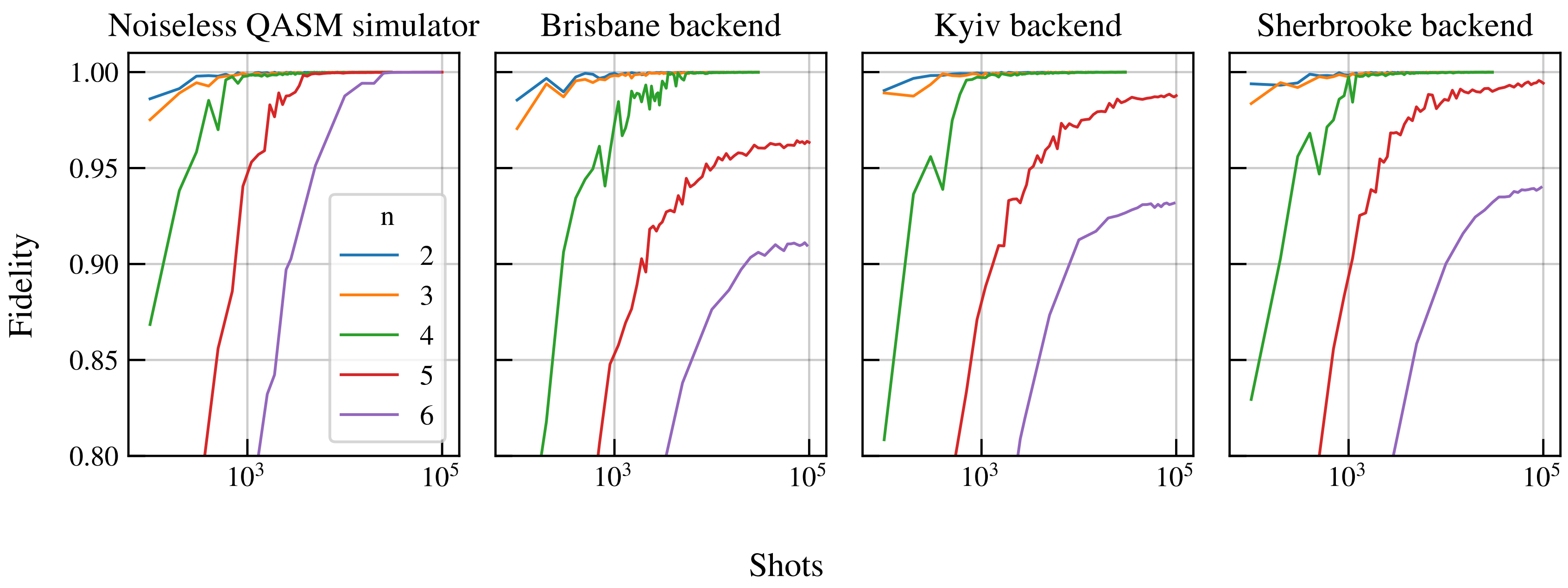
# Getting a physical result

- problem: the obtained matrix is not semi-positive definite...
- solution: tweak it a bit

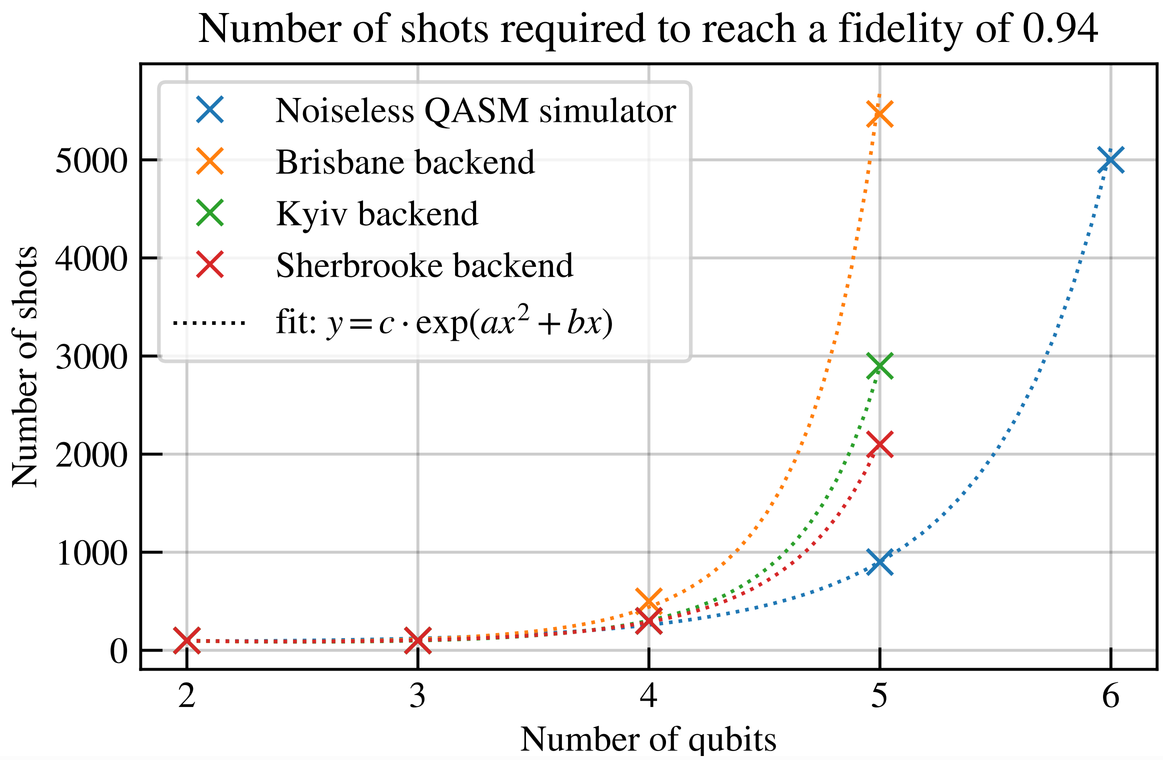
```
def make_it_semi_positive_definite(mat):
    eigvals, v = sy.linalg.eigh(mat)
    if np.all(eigvals>=0):
        return mat
    else:
        eigvals[eigvals<0] = 0
        eigvals = np.real(eigvals)
        result = v@np.diag(eigvals)@v.conj().T
        return result/np.trace(result)
```

- more subtle methods exist :)

# Results


Noiseless QASM simulator
Brisbane backend
Kyiv backend
Sherbrooke backend
Fidelity
1.00
0.95
0.90
0.85
0.80
$10^3$
$10^5$
n
2
3
4
5
6
Shots

# Results


Number of shots required to reach a fidelity of 0.94
Noiseless QASM simulator
Brisbane backend
Kyiv backend
Sherbrooke backend
fit: $y = c \cdot \exp(ax^2 + bx)$
Number of shots
0
1000
2000
3000
4000
5000
Number of qubits
2
3
4
5
6

# Discussion

Concerning my implementation:

- can use other methods of reconstruction that ensure a physically meaningful state
- ML method implemented: poor results because of the Barren plateau?

# Discussion

Concerning my implementation:

- can use other methods of reconstruction that ensure a physically meaningful state
- ML method implemented: poor results because of the Barren plateau?

Concerning the method itself:

- resource needs scale exponentially
- do we really need the full reconstruction of the state?

# Classical shadows

# Main idea

- classical shadows introduced rigorously by HY. Huang, R. Kueng & J. Preskill in 2020

# Main idea

- classical shadows introduced rigorously by HY. Huang, R. Kueng & J. Preskill in 2020
- learn a minimal classical sketch: the classical shadow $S_\rho$
- predict $M$ target functions instead of predicting the full state

# Procedure

$n$ qubits system, $d = 2^n$, fixed ensemble $\mathscr{U}$

1. Rotate the state with a unitary $U$ randomly drawn from $\mathscr{U}$.
2. Perform a measurement in the computational basis: get $\left|\hat{b}\right\rangle \in \{0,1\}^n$.
3. Store efficiently a classical representation of $U^\dagger \left|\hat{b}\right\rangle \left\langle\hat{b}\right| U$.
4. Reconstruct the classical snapshot $\hat{\rho}$ using $\mathscr{M}^{-1}$ that depends on the ensemble.
5. Repeat $N$ times to get $N$ independent classical snapshots of $\rho$:

$$S_\rho(N) = \left\{\hat{\rho}_1 = \mathscr{M}^{-1}\left(U_1^\dagger \left|\hat{b}_1\right\rangle \left\langle\hat{b}_1\right| U_1\right), \ldots, \hat{\rho}_N = \mathscr{M}^{-1}\left(U_N^\dagger \left|\hat{b}_N\right\rangle \left\langle\hat{b}_N\right| U_N\right)\right\}$$

6. Evaluate the target functions via the median of means protocol.

# Set up

- Ensemble: Clifford group
- $\hat{\rho} = \mathcal{M}^{-1}(U^\dagger \left|\hat{b}\right\rangle \left\langle\hat{b}\right| U) = (2^n + 1)U^\dagger \left|\hat{b}\right\rangle \left\langle\hat{b}\right| U - \mathbb{1}$

# Set up

- Ensemble: Clifford group
- $\hat{\rho} = \mathcal{M}^{-1}(U^\dagger \left|\hat{b}\right\rangle \left\langle\hat{b}\right| U) = (2^n + 1)U^\dagger \left|\hat{b}\right\rangle \left\langle\hat{b}\right| U - \mathbb{1}$
- Fidelity

$$F_{GHZ}\left(\hat{\rho}\right) = \mathrm{Tr}\left( \overbrace{|GHZ\rangle\langle GHZ|}^{\text{our observable!}} \hat{\rho} \right)$$
$$= (2^n + 1)\left|\left\langle \hat{b}\,|\,U\,|\,GHZ \right\rangle\right|^2 - 1$$

# Set up

- Ensemble: Clifford group
- $\hat{\rho} = \mathcal{M}^{-1}(U^\dagger \left|\hat{b}\right\rangle \left\langle\hat{b}\right| U) = (2^n + 1)U^\dagger \left|\hat{b}\right\rangle \left\langle\hat{b}\right| U - \mathbb{1}$
- Fidelity

$$F_{GHZ}(\hat{\rho}) = \mathrm{Tr}\Big(\overbrace{|GHZ\rangle\langle GHZ|}^{\text{our observable!}}\hat{\rho}\Big)$$
$$= (2^n + 1)\left|\left\langle \hat{b} | U | GHZ \right\rangle\right|^2 - 1$$

```python
def measurement_to_fidelity(n, outcomes):
    GHZ = init_GHZ(n)
    amplitudes = np.zeros(len(outcomes))
    for i, outcome in enumerate(outcomes):
        rotated_GHZ = GHZ.compose(outcome['clifford'].to_circuit())
        stabilizer_circuit = qi.StabilizerState(rotated_GHZ)
        amplitude =
         ↪  stabilizer_circuit.probabilities_dict_from_bitstring(outcome_bitstring=outcome['bitstring'])
        amplitudes[i] = list(amplitude.values())[0]
    return (2**n+1)*amplitudes-1
```

# Theoritical bounds

- Clifford measurements: size $\sim \mathcal{O}\left(\log(M)\max_i \mathrm{Tr}\left(O_i^2\right)/\varepsilon^2\right)$

# Theoritical bounds

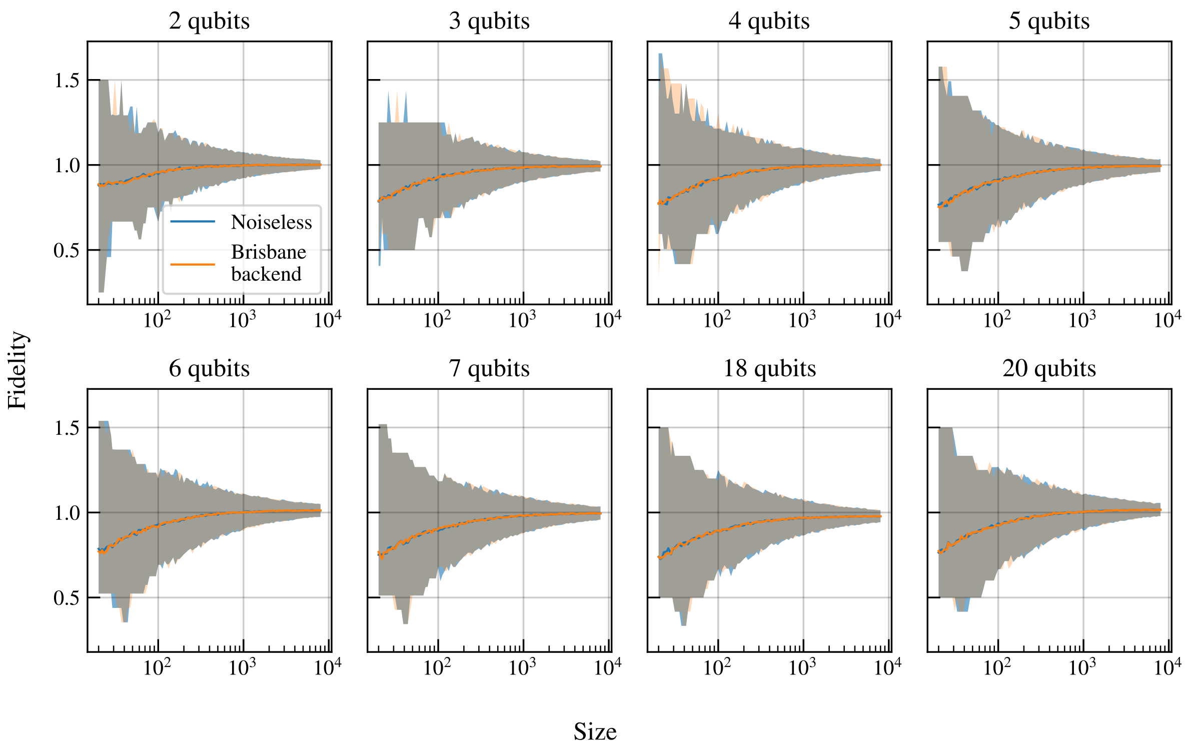
- Clifford measurements: size $\sim \mathcal{O}\left(\log(M)\max_i \mathrm{Tr}\left(O_i^2\right)/\varepsilon^2\right)$
- Pure observables $\implies$ size $\sim \mathcal{O}\left(\log(M)/\varepsilon^2\right)$

# Theoritical bounds

- Clifford measurements: size $\sim \mathcal{O}\left(\log(M) \max_i \mathrm{Tr}\left(O_i^2\right)/\varepsilon^2\right)$
- Pure observables $\implies$ size $\sim \mathcal{O}\left(\log(M)/\varepsilon^2\right)$

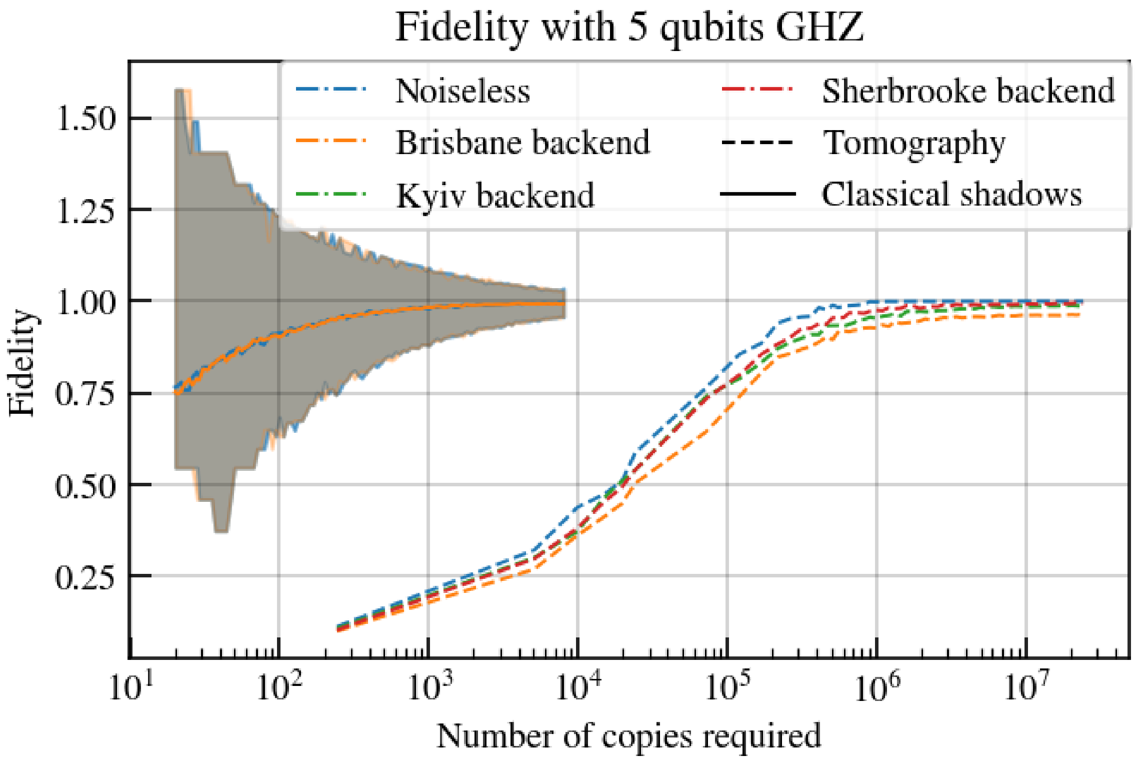
Classical shadows are asymptotically optimal.

# Results



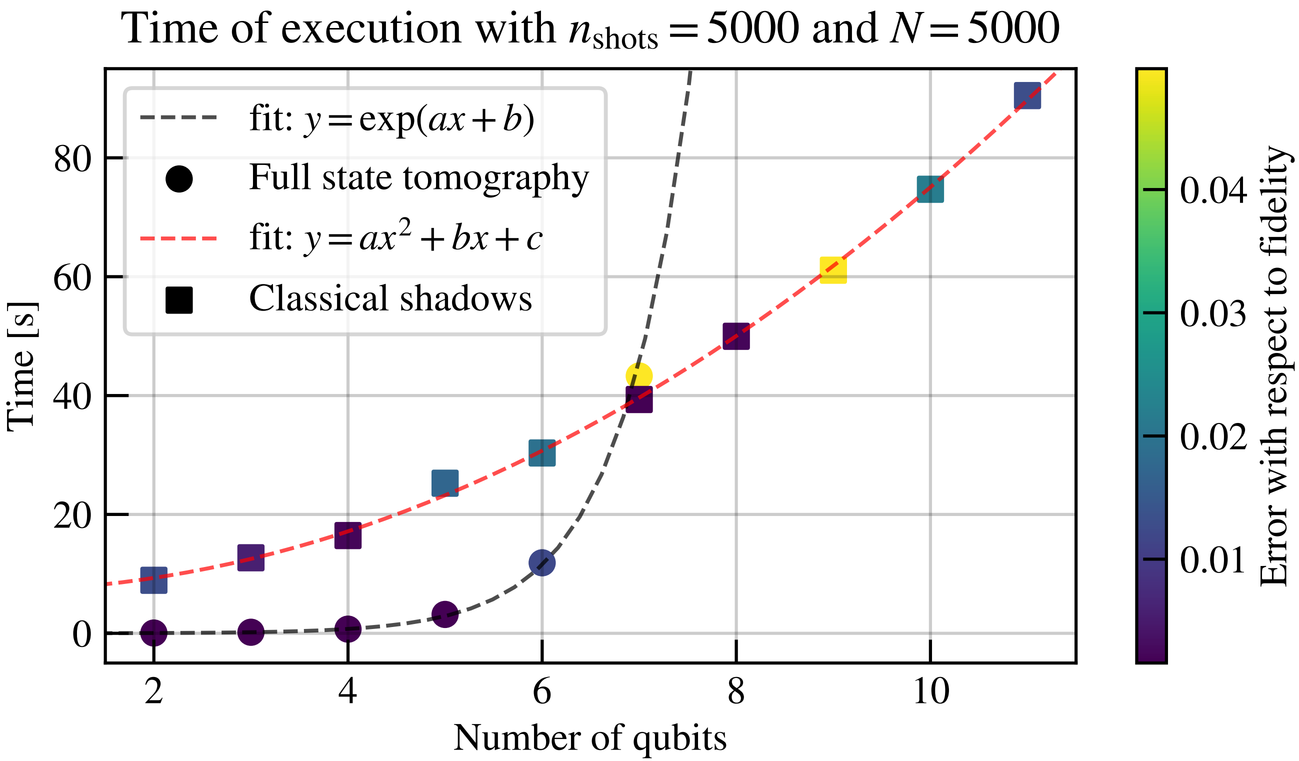

95% confidence interval obtained via bootstrap resampling.

# Results



Fidelity with 5 qubits GHZ
Noiseless
Brisbane backend
Kyiv backend
Sherbrooke backend
Tomography
Classical shadows
Fidelity
1.50
1.25
1.00
0.75
0.50
0.25
$10^1$
$10^2$
$10^3$
$10^4$
$10^5$
$10^6$
$10^7$
Number of copies required

# Results


Time of execution with $n_{\text{shots}} = 5000$ and $N = 5000$
fit: $y = \exp(ax + b)$
Full state tomography
fit: $y = ax^2 + bx + c$
Classical shadows
Time [s]
80
60
40
20
0
2
4
6
8
10
Number of qubits
0.04
0.03
0.02
0.01
Error with respect to fidelity

# State Tomography Using Classical Shadows

Julia Guignon

January 16, 2025 — Ecole Polytechnique Fédérale de Lausanne

# Backup

# Maximum Likelihood approach: reconstruction

```
def multi_qubits_ML_tomography(n, method='Powell', maxit=200, nshots=1000, eps=1e-6,
 ↪  noise=False):
    rhotilde = contruct_rho_lin(n, expectation_values, measurements, valid=True)
    L = np.linalg.cholesky(rhotilde+eps*np.eye(2**n))
    L = L/np.trace(L)
    print('Cholesky done')
    optimization = sy.optimize.minimize(loss_function, get_t_from_mat(L), method=method,
     ↪  args=(expectation_values, measurements))
    opt_t = optimization.x
    print('Optimization done')
    T = parametrization_state_ML(opt_t)
    TdaggerT = T.transpose().conj()@T
    rhotilde = TdaggerT/TdaggerT.trace()
    return rhotilde.todense()
```

$$\rho = \frac{T(t)^\dagger T(t)}{\mathrm{Tr}\big(T(t)^\dagger T(t)\big)} \qquad T(t) = \begin{pmatrix} t_1 & 0 & 0 & 0 \\ t_5 + it_6 & t_2 & 0 & 0 \\ t_{11} + it_{12} & t_7 + it_8 & t_3 & 0 \\ t_{15} + it_{16} & t_{13} + it_{14} & t_9 + it_{10} & t_4 \end{pmatrix}$$

# Maximum Likelihood approach: loss function

```
def loss_function(t, expect_val_measured, measurements):
    rho_t = parametrization_state_ML(t)
    rho_t_expectation_value = lambda x: get_expectation_value_from_str(rho_t, x)
    expectation_values_parametrized = np.array(list(map(rho_t_expectation_value,
    ↪   measurements.keys())))
    expect_val_measured = np.array(list(expect_val_measured.values()))
    return np.sum((expectation_values_parametrized-expect_val_measured)**2)
```

# Bootstrap resampling

```
def bootstrap_confidence_interval(data, size, K, iterations=1000):
    means = np.zeros(iterations)

    for i in range(iterations):
        bootstrap_sample = np.random.choice(data, size=size, replace=True)
        means[i] = median_of_means_fidelity(bootstrap_sample, K)

    lower_bound = np.percentile(means, 2.5)
    upper_bound = np.percentile(means, 97.5)
    mu = np.mean(means)
    return [mu, mu-lower_bound, upper_bound-mu]
```

# Median of means

```
def median_of_means_fidelity(cl_shadow, K):
    N = len(cl_shadow)
    fidelities = np.zeros(K)
    for k in range(1, K+1):
        fidelities[k-1] = np.mean(cl_shadow[(k-1)*N//K:k*N//K+1])
    return np.median(fidelities)
```

# Classical shadows via Pauli strings ensemble

- Ensemble: Pauli string with weight $k$
- $\hat{\rho}^{(m)} = \frac{1}{K} \sum_{i=1}^{K} \bigotimes_{n=1}^{N} \left( 3 \left( U_n^{(m)} \right)^{\dagger} \left| b_n^{(m,k)} \right\rangle \left\langle b_n^{(m,k)} \right| U_n^{(m)} - \mathbb{1} \right)$
- $N \sim \mathcal{O}(3^k \log(M)/\varepsilon^2)$